

IBM WebSphere Application Server Liberty Profile Guide for Developers

Miho Hachitani
Catalin Mierlea
Pete Neergaard
Grzegorz Smolko



WebSphere



International Technical Support Organization

**IBM WebSphere Application Server Liberty Profile
Guide for Developers**

July 2015

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

Third Edition (July 2015)

This edition applies to WebSphere Application Server V8.5.5.6.

© Copyright International Business Machines Corporation 2012, 2013, 2015. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
IBM Redbooks promotions	ix
Preface	xi
Authors	xi
Now you can become a published author, too!	xiii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiv
Chapter 1. An introduction to the Liberty profile	15
1.1 Liberty profile overview	16
1.1.1 Programming models	16
1.1.2 Supported development environments	17
1.1.3 Application development and deployment tools	17
1.1.4 Additional resources	18
1.2 Simplified configuration	19
1.2.1 Server definition	19
1.2.2 Server configuration using the server.xml file	19
1.2.3 Bootstrap properties	20
1.2.4 Portable configuration using variables	20
1.2.5 New configuration types and validation	23
1.2.6 Encoding passwords	23
1.2.7 Shared configuration using includes	24
1.2.8 Dynamic configuration updates	25
1.3 Runtime composition with features and services	26
1.3.1 Feature management	26
1.3.2 Automatic service discovery	27
1.4 Frictionless application development	28
1.4.1 Quick start using dropins	28
1.4.2 Configuration-based application deployment	28
1.4.3 Using loose configuration for applications	29
1.4.4 Configuring an application's context root	30
1.4.5 Compatibility with WebSphere Application Server	30
1.5 Product extensions	30
1.6 Overview of Java EE 7	32
1.6.1 Java EE 7 Web Profile	32
1.6.2 Java EE 7 Full Platform	33
Chapter 2. Installation	37
2.1 Installing the WebSphere developer tools	38
2.1.1 Installation from Eclipse Marketplace	38
2.1.2 Installation from the WASdev community site	40
2.1.3 Installation from downloaded installation files	42
2.2 Installing the Liberty profile	43
2.2.1 Installation using the Liberty profile developer tools	43
2.2.2 Installation using the command line	49
2.2.3 Installation on z/OS	51

2.2.4	The Liberty profile runtime environment and server directory structure	52
2.3	Configuring the server runtime JDK	55
2.3.1	Defining the JRE from within workbench	55
2.3.2	Configuring the system JRE	56
2.3.3	Using server.env to define the JRE.	57
2.3.4	Specifying JVM properties	57
2.4	Starting and stopping a Liberty profile server	57
2.4.1	Starting and stopping the server by using the command line	58
2.4.2	Starting and stopping the server from the workbench.	58
Chapter 3. Developing and deploying web applications		59
3.1	Developing applications using the Liberty profile developer tools.	60
3.1.1	Using the tools to create a simple servlet application	60
3.1.2	Developing and deploying a JSP application	62
3.1.3	Developing and deploying a JSF application	64
3.1.4	Developing and deploying JAX-RS applications.	70
3.1.5	Using Context and Dependency Injection in web applications with the Liberty profile developer tools	75
3.1.6	Developing JAX-WS web services applications with the Liberty profile developer tools	78
3.1.7	Developing WebSocket applications with the Liberty profile developer tools	82
3.1.8	Debugging applications with the Liberty profile developer tools	84
3.2	Developing outside the Liberty profile developer tools	86
3.2.1	Feature enablement	87
3.2.2	Dynamic application update	87
3.2.3	Common development configuration	88
3.2.4	Dynamic configuration	88
3.2.5	API JAR files	89
3.2.6	Debugging applications.	90
3.2.7	Using Maven to automate tasks for the Liberty profile	90
3.2.8	Using Ant to automate tasks for the Liberty profile	93
3.3	Controlling class visibility in applications.	94
3.3.1	Using shared libraries in applications	94
3.3.2	Creating a shared library in the Liberty profile developer tools.	94
3.3.3	Creating a shared library outside of the tools	95
3.3.4	Using libraries to override Liberty profile server classes.	96
3.3.5	Global libraries	96
3.3.6	Using a classloader to control API visibility.	96
Chapter 4. Iterative development of OSGi applications		99
4.1	Introduction to OSGi applications in Liberty profile	100
4.2	Developing OSGi applications in the Liberty profile developer tools.	100
4.2.1	Using the tools to build an OSGi application.	101
4.2.2	Using the tools to deploy and test an OSGi application	106
4.2.3	Adding an OSGi web application bundle	107
4.2.4	Deploying an OSGi application to Liberty in the cloud	108
4.3	Developing OSGi applications outside Liberty profile developer tools	110
4.3.1	Building and deploying OSGi applications outside of the tools.	110
4.3.2	Using Maven to automate OSGi development tasks	111
4.3.3	Using Ant to automate OSGi development tasks	111
Chapter 5. Developing enterprise applications with Liberty profile.		113
5.1	Data access in the Liberty profile	114
5.1.1	Accessing data using a data source and JDBC	114

5.1.2	Developing JPA applications	122
5.1.3	Data access with noSQL database	131
5.2	Developing Enterprise JavaBeans applications	141
5.2.1	What's new in developing EJB applications for Liberty profile	141
5.2.2	Developing applications using local EJB	141
5.2.3	Developing applications using remote EJB	144
5.2.4	Developing applications using timers	152
5.3	Developing Java Message Service applications	158
5.3.1	What's new in JMS 2.0 API?	159
5.3.2	Developing applications using JMS 2.0 API	159
5.3.3	Developing a message-driven bean (MDB) application	166
5.4	Developing applications using asynchronous and concurrent features	168
5.4.1	Developing using @Asynchronous in servlets and filters	169
5.4.2	Developing using @Asynchronous in session beans	171
5.4.3	Developing using Concurrent API	173
5.5	Developing applications using JavaMail	177
5.5.1	Writing, testing, and deploying the JavaMail sample application	177
Chapter 6.	Configuring application security	181
6.1	Enabling SSL	182
6.1.1	Configuration using the WebSphere developer tools	182
6.1.2	Configuration using the command line	185
6.2	HTTPS redirect	185
6.3	Form login	187
6.3.1	Defining the basic user registry	187
6.3.2	Updating an application to support Form Login	190
6.3.3	Defining bindings between the application and the server	193
6.4	Securing EJB applications	195
6.5	Securing JMS applications	196
6.5.1	Setting up user authentication	196
6.5.2	Setting up user authorization	199
6.6	JAX-WS security	200
6.7	Securing NoSQL applications	203
6.7.1	Securing MongoDB applications	203
6.7.2	Securing CouchDB applications	203
6.8	Authenticating users in Liberty profile	204
6.8.1	User registries for WebSphere Liberty profile	204
6.8.2	Custom authentication methods	205
Chapter 7.	Serviceability and troubleshooting	209
7.1	Logs and trace	210
7.1.1	Inspecting the output logs and trace	210
7.1.2	Configuration of an additional trace	211
7.2	Server memory dump	213
7.3	MBeans and JConsole	213
7.4	OSGi Debug console	217
7.5	Event logging feature	218
7.6	Slow request detection and hung request detection capabilities	219
7.7	Timed Operations feature	220
Chapter 8.	From development to production	221
8.1	Configuring a server for production use	222
8.1.1	Turning off application monitoring	222
8.1.2	Turning off configuration file monitoring	222

8.1.3	Generating a web server plug-in configuration	222
8.2	Using the package utility	223
8.2.1	Packaging a Liberty profile server by using the WebSphere developer tools . . .	223
8.2.2	Packaging a Liberty profile server from a command prompt.	224
8.2.3	Using the Job Manager to package and distribute Liberty profile servers	224
8.2.4	Using the Liberty collective to distribute Liberty profile servers	224
8.3	Moving an application to the full profile	224
8.3.1	Programming model differences between full profile and Liberty profile.	225
8.3.2	Configuration differences between full profile and Liberty profile	225
8.4	Using the Liberty profile on z/OS	228
8.4.1	IBM z/OS Connect	230
8.4.2	WebSphere optimized local adapters for z/OS	232
8.4.3	Disabling z/OS operator console command handling	233
Chapter 9. Developing and deploying custom features		235
9.1	Considerations for creating custom features.	236
9.2	Defining a custom feature	237
9.2.1	Elements of a feature	237
9.2.2	Visibility constraints for features, packages, and services	239
9.2.3	Subsystem content: Writing a minify-compatible feature	242
9.2.4	Using the tools to create a custom feature	243
9.2.5	Creating a Liberty feature manually	249
9.2.6	Automatic provisioning: Creating an auto-feature	251
9.2.7	Packaging native code in your bundles	251
9.2.8	Packaging features for delivery	252
Appendix A. Additional material		257
	Locating the web material	257
	Using the web material.	257
	Downloading and extracting the web material	258
	Importing the Liberty profile developer tools projects	258
	Using the compressed Derby database files.	258
Related publications		259
	Online resources	259
	Help from IBM	259

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	IBM z™	Rational®
Bluemix™	IMS™	Redbooks®
CICS®	MVS™	Redbooks (logo)  ®
DB2®	Passport Advantage®	WebSphere®
IBM®	PureApplication®	z/OS®

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

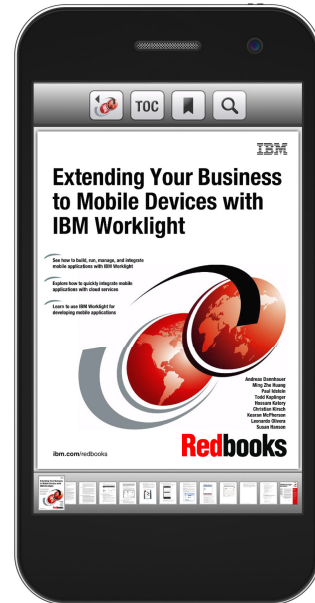
UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Find and read thousands of IBM Redbooks publications

- ▶ Search, bookmark, save and organize favorites
- ▶ Get up-to-the-minute Redbooks news and announcements
- ▶ Link to the latest Redbooks blogs and videos

Get the latest version of the **Redbooks Mobile App**



Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!



ibm.com/Redbooks
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

IBM® WebSphere® Application Server V8.5 includes a Liberty profile, which is a highly composable, dynamic application server profile. It is designed for two specific use cases: Developer with a smaller production run time, and production environments. For a developer, it focuses on the tasks that a developer does most frequently and makes it possible for the developer to complete those tasks as quickly and as simply as possible. For production environments, it provides a dynamic, small footprint run time to be able to maximize system resources.

This IBM Redbooks® publication provides you with information to effectively use the WebSphere Application Server V8.5 Liberty profile along with the WebSphere Application Server Developer Tools for Eclipse, for development and testing of web applications that do not require a full Java Platform. It provides a quick guide on getting started, providing a scenario-based approach to demonstrate the capabilities of the Liberty profile along with the developer tools. This provides a simplified, but comprehensive, application development and testing environment.

The intended audience for this book is developers of web and Open Services Gateway initiative (OSGi) applications who are familiar with web and OSGi application concepts.

This book has been updated to reflect the new features in WebSphere Application Server.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.



Miho Hachitani is an advisory specialist in the technical support team for WebSphere Application Server. She has over 13 years of experience at the WebSphere Application Server support organization in IBM, including design of topologies, scalability, high availability, performance, and administration.

She delivered customer projects that adopt the Liberty profile, and tested many kinds of use cases of Liberty profile, including Liberty collective.



Catalin Mierlea is a Middleware Software Specialist in Technical Support Services of Global Technologies Services, IBM Romania. Catalin joined IBM in March 2012 and has 10 years experience with IBM middleware software. His areas of expertise include WebSphere products, SOA, and software architecture. He specializes in WebSphere Application Server, WebSphere Portal Server, and WebSphere Business Process Manager. Catalin has a BS in Automation Control and Computers, an MS in Integrated Informatics Systems, IBM WebSphere products certifications, and competencies in different Oracle and Microsoft technologies. He has extensive industry knowledge and hands-on project experience in the banking and the public sector.



Pete Neergaard is a Certified IT Specialist working as a Course Developer and Instructor in the WebSphere Application Server area. He has been at IBM for 18 years, working for the WebSphere Education team with focus on areas including WebSphere Application Server, Intelligent Management, Security, and Mobile. Previously, he worked at Carnegie Mellon University as a Research Systems Programmer. He holds a Bachelor's degree in Computer Science and Applied Math from Carnegie Mellon University. He currently lives and works in Pittsburgh, Pennsylvania.



Grzegorz Smolko is a Certified IT Specialist with IBM Poland in Warsaw. Grzegorz has been working for IBM for more than twelve years, mostly in IBM Software Services for WebSphere. Lately, he is a member of the Worldwide WebSphere Competitive Migration Team, helping customers to migrate their application from various platforms to WebSphere Application Server and WebSphere Liberty profile. Before joining IBM, he worked for software house companies in Poland as a Java developer and architect. His areas of expertise include Java; Java Platform, Enterprise Edition; and WebSphere. He holds certifications from Oracle and IBM in Java and WebSphere technologies. He has a Master's degree in Computer Science from the Warsaw University of Technology, Poland.

This project was led by:

- ▶ Margaret Ticknor, a Redbooks Project Leader in the Raleigh Center. She primarily leads projects about WebSphere products and IBM PureApplication® System. Before joining the

ITSO, Margaret worked as an IT specialist in Endicott, NY. Margaret attended the Computer Science program at State University of New York at Binghamton.

Thanks to the following people for their contributions to this project:

- ▶ Paul W. Bennett, IBM Systems, Middleware, WebSphere Development, IBM US
- ▶ Kihup Boo, IBM Systems, Middleware, IBM Canada
- ▶ Robert Haimowitz, DST Poughkeepsie, Enablement for IBM z™ Systems & z/OS®
- ▶ Alex Mulholland, STSM WebSphere Application Server Liberty Runtime Architect, IBM US
- ▶ Alasdair Nottingham, WebSphere Application Server Liberty Profile Development Lead, IBM Hursley
- ▶ Gary R. Picher, Software Engineer, WebSphere for z/OS, IBM US
- ▶ Preethi R. Sulkunte, IBM Systems, Middleware, IBM India Pvt Ltd

Thanks to the authors of the previous editions of IBM WebSphere Application Server Liberty Profile Guide for Developers: Brent Daniel, Alex Mulholland, Erin Schnabel, Kevin Smith.

Thanks to the following people for their support of this project:

- ▶ Deana Coble, IBM Redbooks Technical Writer
- ▶ Lindamay Patterson, IBM Redbooks Technical Writer
- ▶ Tamikia Lee, IBM Redbooks Residency Administrator

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099

2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<https://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



An introduction to the Liberty profile

The IBM WebSphere Application Server V8.5 Liberty Profile is a composable, dynamic application server environment that supports development and testing of web applications. Recent updates add not just Java EE Web Profile support, but also Java EE Full Platform support. The profile is fully supported for production use, but is designed to work alongside WebSphere Application Server Developer Tools for Eclipse to provide a simplified, but comprehensive, application development and testing environment.

This guide is for developers of web and Open Services Gateway initiative (OSGi) applications. It assumes familiarity with web and OSGi application concepts, but does not assume familiarity with WebSphere Application Server resources or products. This guide refers to other IBM Redbooks publications and the WebSphere Application Server V8.5 IBM Knowledge Center for additional information.

The topics in this chapter provide an overview of Liberty profile characteristics that are useful for iterative and collaborative application development:

- ▶ Liberty profile overview
- ▶ Simplified configuration
- ▶ Runtime composition with features and services
- ▶ Frictionless application development
- ▶ Product extensions
- ▶ Overview of Java EE 7

1.1 Liberty profile overview

The Liberty profile is a simplified, lightweight development and application runtime environment that has the following characteristics:

- ▶ Simple to configure. Configuration is read from an XML file with text-editor-friendly syntax.
- ▶ Dynamic and flexible. The run time loads only what your application needs and recomposes the run time in response to configuration changes.
- ▶ Fast. The server starts in under 5 seconds with a basic web application.
- ▶ Extensible. The Liberty profile provides support for user and product extensions, which can use System Programming Interfaces (SPIs) to extend the run time.

The Liberty profile is built by using OSGi technology and concepts. The fit-for-purpose nature of the run time relies on the dynamic behavior inherent in the OSGi Framework and Service Registry. As bundles are installed to or uninstalled from the framework, the services that each bundle provides are added or removed from the service registry. The addition and removal of services similarly cascades to other dependent services. The result is a dynamic, composable run time that can be provisioned with only what your application requires and responds dynamically to configuration changes as your application evolves.

1.1.1 Programming models

The Liberty profile supports a subset of the Java EE 6 and Java EE 7 stacks. The following list notes the supported technologies that are supported by all versions of the Liberty profile:

- ▶ Java Servlet 3.1
- ▶ JavaServer Faces (JSF) 2.2
- ▶ JavaServer Pages (JSP) 2.3
- ▶ Java Expression Language 3.0
- ▶ Standard Tag Library for JavaServer Pages (JSTL) 1.2
- ▶ Bean Validation 1.1
- ▶ Java Persistence API (JPA) 2.1
- ▶ Common Annotations for the Java Platform 1.1
- ▶ Java Transaction API (JTA) 1.2
- ▶ Java Database Connectivity (JDBC) 4.1
- ▶ Java API for RESTful Web Services (JAX-RS) 2.0
- ▶ Java API for RESTful Web Service clients (JAX-RS Client) 2.0
- ▶ Java API for XML Processing (JAXP) 1.3
- ▶ Java Management Extensions (JMX) 2.0
- ▶ JavaBeans Activation Framework (JAF) 1.1
- ▶ Streaming API for XML (StAX) 1.0
- ▶ Enterprise Java Beans Lite (EJB Lite) 3.2
- ▶ Contexts and Dependency Injection 1.2

The *WebSphere Application Server Liberty Core edition* is a separate offering that is available for production use that is based on the Liberty profile. The Liberty Core edition extends the technologies that were listed previously to support the following Java EE 6 Web Profile specifications:

- ▶ EE Concurrency Utilities 1.0
- ▶ JavaMail 1.5

In other editions, such as WebSphere Application Server and WebSphere Application Server Network Deployment, the Liberty profile supports the following technologies in addition to those contained in Liberty Core:

- ▶ Java EE 7 Full Platform
- ▶ Java API for XML-based Web Services (JAX-WS) 2.2
- ▶ Java Architecture for XML Binding (JAXB) 2.2
- ▶ SOAP with Attachments API for Java (SAAJ) 1.3
- ▶ Java Message Service API (JMS) 2.0
- ▶ Enterprise JavaBeans 3.1 3.2 - Message-Driven beans

The Liberty profile supports OSGi applications in all editions. The following list shows supported technologies for OSGi applications (with a reference to the specification where appropriate):

- ▶ Web Application Bundles (OSGi R4.2 Enterprise, Chapter 128)
- ▶ Blueprint Container (OSGi R4.2 Enterprise, Chapter 121)
 - Blueprint Transactions
 - Blueprint Managed JPA
- ▶ JNDI (OSGi R4.2 Enterprise, Chapter 126)
- ▶ OSGi application of Java EE technologies that are supported by the profile

A complete list of the technologies that are supported by the Liberty profile can be found at:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/rwlp_prog_model_support.html

1.1.2 Supported development environments

Both Liberty and full profile WebSphere Application Server support a broad list of operating systems: IBM AIX®, HP, IBM i, Linux, Solaris, Windows, and z/OS. The Liberty profile is also supported, for development, on Mac OSX.

The Liberty profile server runs on Java Version 6 or later regardless of the vendor that provides it.

The list of supported operating systems and requirements for the Liberty profile can be found at the following link:

<http://www-01.ibm.com/support/docview.wss?uid=swg27038218>

1.1.3 Application development and deployment tools

IBM provides two development tools that are designed for use with WebSphere Application Server applications and include tools that are specifically suited for use with the Liberty profile:

- ▶ IBM Rational® Application Developer for WebSphere Software V9

IBM Rational Application Developer for WebSphere Software V9 provides a development environment for building applications that run on WebSphere Application Server. This tool supports all Java EE artifacts that are supported by WebSphere Application Server, such as servlets, JavaServer Pages (JSP), JavaServer Faces (JSF), Enterprise JavaBeans (EJB), Extensible Markup Language (XML), SIP, Portlet, and web services. It also includes integration with the OSGi programming model. The workbench contains wizards and editors that help build standards-compliant, business-critical Java EE, Web 2.0, and service-oriented architecture (SOA) applications. Code quality tools help teams find and correct problems before they escalate into expensive problems. Rational Application

Developer for WebSphere Software can be used to develop applications for both the full profile and the Liberty profile.

For more information about Rational Application Developer for WebSphere Software V9, see the following website:

<http://www.ibm.com/software/awdtools/developer/application>

- ▶ WebSphere Application Server Developer Tools for Eclipse

The IBM WebSphere developer tools V8.5.5 provides a development environment for developing, assembling, and deploying Java EE, OSGi, Web 2.0, and Mobile applications, and supports multiple versions of WebSphere Application Server. When combined with Eclipse SDK and Eclipse Web Tools Platform, WebSphere developer tools provides a lightweight environment for developing Java EE applications.

WebSphere Application Server Developer Tools for Eclipse is a no additional charge edition for developer desktop and includes Eclipse adapters. With Version 8.5.5, WebSphere Application Server and WebSphere Application Server Developer Tools for Eclipse editions are provided at no additional charge for developer desktops and are supported under production runtime licenses.

Although not as rich in features as Rational Application Developer for WebSphere Software, this tool is an attractive option for developers using both the Liberty profile and the full profile.

For more information about WebSphere developer tools and access to the tool, see the following website:

https://www.ibm.com/developerworks/community/blogs/wasdev/entry/downloads_final_releases?lang=en

WebSphere developer tools: In this book, the development and deployment processes that are described are applicable to either tool. We use the generic term *WebSphere developer tools* as an inclusive term for both WebSphere Application Server Developer Tools for Eclipse and Rational Application Developer for WebSphere Software.

1.1.4 Additional resources

There are additional resources available to support development with the Liberty profile. The following list notes developer-community focused resources:

- ▶ WASdev community

<http://wasdev.net>

The WASdev community is a hub for information about developing applications for WebSphere Application Server, and using the Liberty profile in particular. Articles, podcasts, videos, and samples are refreshed regularly as new technology is made available through early access programs or new product releases.

- ▶ WASdev forum

<https://developer.ibm.com/wasdev>

The WASdev forum provides an opportunity for users of WebSphere Application Server and the Liberty profile to interact with each other and with the IBM developers working on the products.

- ▶ Stack Overflow tags

<http://stackoverflow.com/questions/tagged/websphere-liberty>

Stack Overflow is a Q&A site that has no cost to use. It allows users to ask and answer questions with built-in incentives to reward high-quality answers. Use the `websphere` or `websphere-liberty` tags to ask or answer questions about WebSphere Application Server and the Liberty profile.

1.2 Simplified configuration

The Liberty profile runtime environment is composed of a set of OSGi bundles. Any bundle containing configurable resources also contains metadata describing the configuration that those resources accept or require using the OSGi Metatype service. A generic description of a metatype service is a bundle containing within it an XML schema describing each resource's configuration attributes, types, and value bounds. The configuration metadata within each bundle will also assign appropriate default values, and can construct default instances where appropriate. With functional defaults already in place, you only need to specify the values you want to customize.

1.2.1 Server definition

The Liberty profile supports multiple servers using the same installation. Each server is defined in its own directory, where the directory name is the server name. New servers are easily created using the command line or the WebSphere developer tools, but manual creation also works. Using the WebSphere developer tools to create a server is described in 2.2, "Installing the Liberty profile" on page 43.

A server definition requires only one configuration file, `server.xml`, to define a server. The `server.xml` file must contain a server element. If there are no configuration changes, default values suffice. The server element can be empty, as shown in Example 1-1.

Example 1-1 Server.xml file

```
<server description="optional description" />
```

The empty definition in Example 1-1 starts only the core kernel. Additional function is enabled using features. Features are described in 1.3.1, "Feature management" on page 26.

1.2.2 Server configuration using the `server.xml` file

The `server.xml` file has a simple XML format that can be edited using your favorite text editor. Though there is no requirement to use a GUI to edit your server's configuration, the WebSphere developer tools provide an enhanced editor for use in an Eclipse IDE. This editor has context-sensitive help, wizard-style value selection, and built-in configuration validation. The enhanced editor also allows direct viewing and editing of the XML source.

Figure 1-1 on page 20 shows the design and source tabs of the enhanced server.xml editor in the WebSphere developer tools.

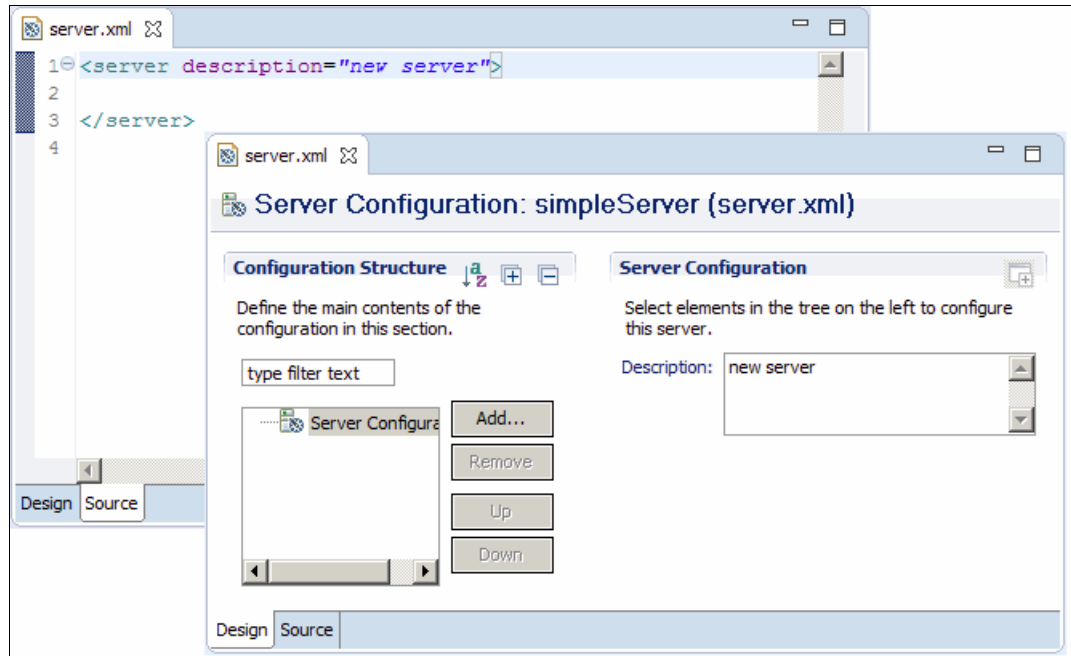


Figure 1-1 The source and design tabs show different views of the server's configuration

The subsequent chapters in this publication provide examples using the enhanced editor to make server configuration updates. If you follow the examples, you might find it interesting to switch to the source view. There, you can observe changes in the server.xml file after using the enhanced editor to modify the server's configuration.

1.2.3 Bootstrap properties

There is another configuration file, which is used by the Liberty profile server when initializing the OSGi framework and platform core, called bootstrap.properties. The bootstrap.properties file is an optional peer of the server.xml file containing simple text properties as key=value pairs. Bootstrap properties are generally used to influence early server start behavior or to define machine-specific properties. The bootstrap properties file is not created by default.

1.2.4 Portable configuration using variables

The Liberty profile configuration manager resolves variables when processing configuration information. Using configuration variables allows environment-specific attributes to be removed from the bulk of the server's configuration. This isolates general aspects of server or application configuration from environment-specific elements such as host names, port numbers, or file system locations.

The Liberty profile provides several predefined variables to help make server configurations portable. The following list demonstrates syntax and briefly summarizes the supported location variables:

wlp.install.dir This is the profile's installation directory.

wlp.user.dir	This directory is the root of the tree containing user configuration. By default, this directory is a child of the installation directory, <code>\${wlp.install.dir}/usr</code> . It can be configured to an alternative location using an environment variable when the server is started.
wlp.server.name	This is the name of the server.
server.config.dir	The server configuration directory contains the <code>server.xml</code> file and any other optional configuration files that are used to customize the server's environment. The name of this directory is used as the server's name. By default, this directory is a child of the user directory called <code>\${wlp.user.dir}/servers/serverName</code> .
server.output.dir	The server output directory contains resources that are generated by the server, such as logs. The Liberty profile supports shared configurations, for example, multiple running server instances sharing the configuration files. Shared configurations are supported by allowing the instance-specific output directory to be moved using an environment variable when the server is started. By default, the server output directory is the same as the server's configuration directory.
shared.app.dir	Shared user applications directory, <code>\${wlp.user.dir}/shared/apps</code> .
shared.config.dir	Shared user configuration directory, <code>\${wlp.user.dir}/shared/config</code> .
shared.resource.dir	Shared user resources directory, <code>\${wlp.user.dir}/shared/resources</code> .

When specifying file or resource locations, use the most specific location variable that applies. Being specific helps to ensure that your server configuration can be propagated to other environments (a fellow developer's notebook, a dedicated test automation framework, a distributed production environment) without requiring changes to the `server.xml` file.

Example 1-2 shows a simple `bootstrap.properties` file that defines custom variables for machine-specific host and ports, and a common file location that is based on one of the built-in location variables.

Example 1-2 A simple bootstrap.properties file

```
appCacheFile=${server.output.dir}/application/cache
hostName=*
httpPort=9080
httpsPort=9443
```

Figure 1-2 on page 22 shows an example of replacing values in a server.xml file with variables defined in the bootstrap.properties file from Example 1-2.

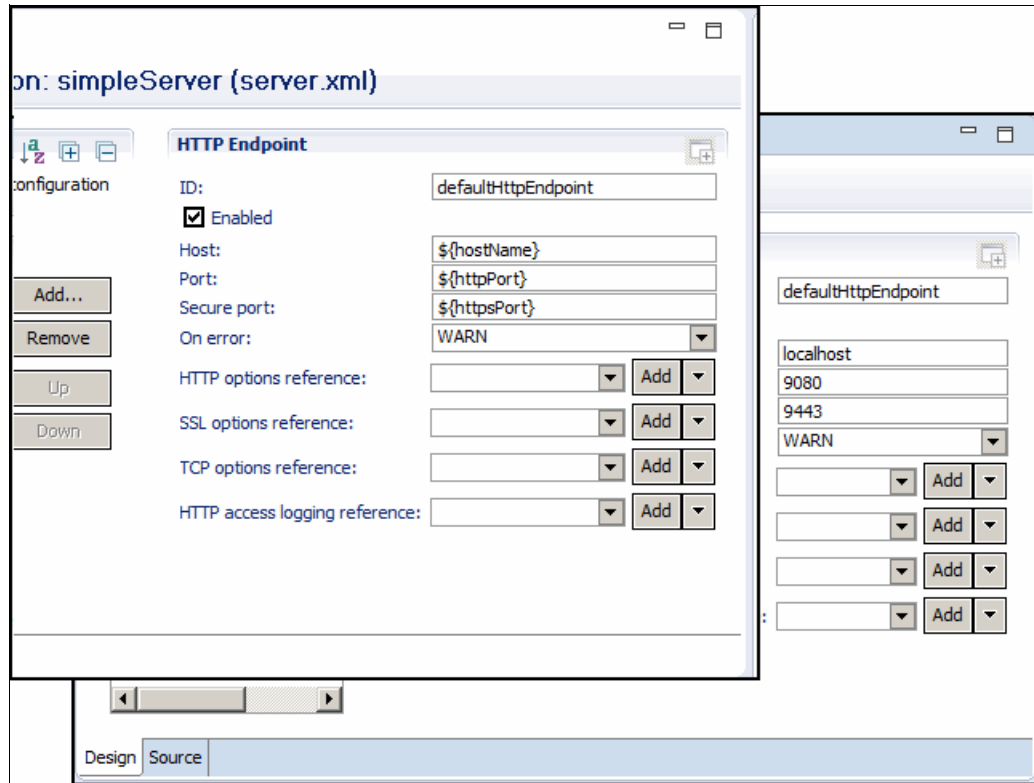


Figure 1-2 Using variables in the enhanced editor

The enhanced editor provides content assist for variables that are defined in either the bootstrap.properties or server.xml files. Variables are filtered based on the field being configured. If the field is for a number (integer, long, and so on), the content assist list will show variables only with numeric values. This context-sensitive variable filtering is shown in Figure 1-3.

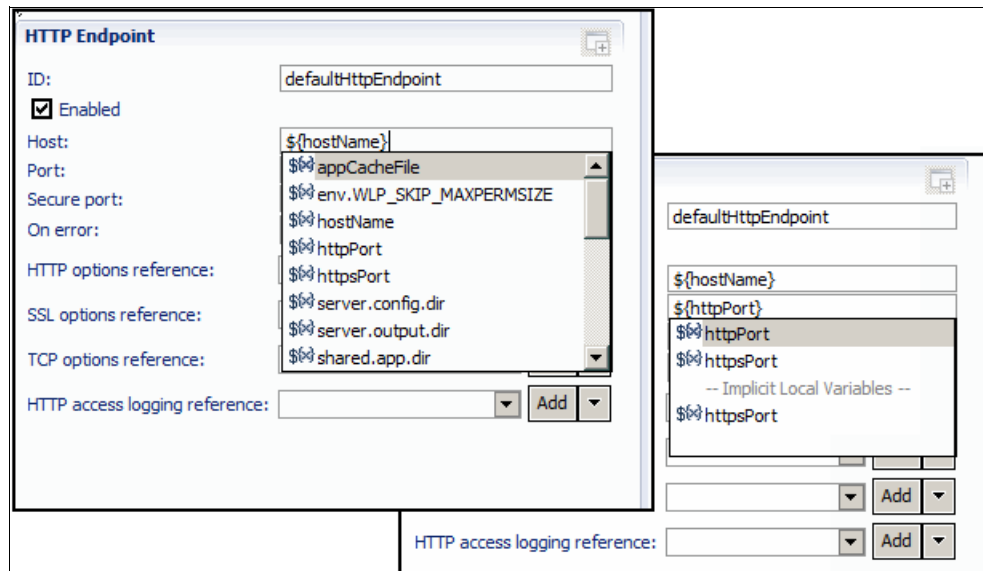


Figure 1-3 Content assist will filter-defined variables

1.2.5 New configuration types and validation

The simple XML format that is used in `server.xml` uses intuitive, natural types to make configuring the server more natural. For example, durations and timeouts can be specified using strings containing units of time (specify `1m30s` for 1 minute and 30 seconds).

The server is tolerant of configuration mistakes where possible. If the initial configuration for the server is malformed, the server will not start, but indicates where the syntax error occurred to help with correcting the problem. If the configuration for a running server is updated with invalid syntax, the change is not applied, and the server continues to run. It also displays a message indicating where the configuration problem was so it can be corrected. Subsequent updates to correct the error are applied to the server without requiring a server restart. A configuration that is provided for unknown or inactive elements is ignored. You do not have to remove configuration for features that are disabled.

The following website provides an index describing all configurable attributes of the run time:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/autodita/rwlp_metatype_4ic.html?cp=SSAW57_8.5.5%2F1-0-2-1-0

1.2.6 Encoding passwords

Passwords can be specified as encoded strings to alleviate concerns about using plain text passwords in configuration files. There are two ways to create and specify an encoded password:

- ▶ The command line. You can use the `securityUtility` script in the `${wlp.install.dir}/bin` directory:

```
securityUtility encode myPassword
```

This returns an encoded string, starting with “`{xor}...`”, to be copied verbatim into your `server.xml` file.

Tip: The `securityUtility` script can be used to produce passwords that are XOR-encoded, hash-encoded, or aes-encrypted. Run the following `help` command for more information about the new encoding options:

```
securityUtility help encode
```

- ▶ The enhanced editor. When the editor is used to configure passwords, a standard password dialog box performs the same encoding step and sets the attribute value to the resulting encoded string.

1.2.7 Shared configuration using includes

The Liberty profile also supports composed configuration. It is possible for the `server.xml` file to include other files. Included files are specified using the `<include.../>` directive, and must use the `server.xml` file's syntax. Each included file must also contain a `<server.../>` element. Figure 1-4 provides a basic representation of how the configuration manager works with configuration coming from various sources. Each bundle in the run time provides configuration defaults and configuration metadata. The configuration manager reads this information from each bundle and merges it with information read from the `server.xml` file and any included files. The configuration manager will stop processing, with an error, if any required configuration files cannot be found.

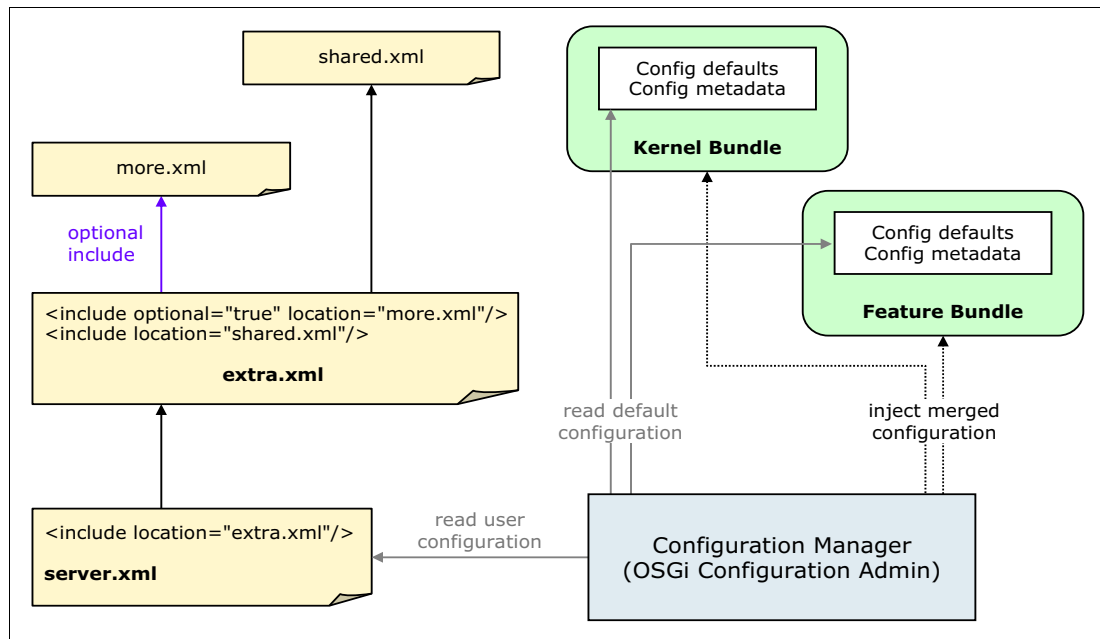


Figure 1-4 The configuration manager processing configuration metadata

By using variables and included files, a wide variety of configuration patterns becomes possible. Configuration can be shared across servers or persisted in a source control repository at varying levels of granularity. For example, a configuration that is associated with a data source can be collected into a single XML file that is reused (as an included resource) across multiple server configurations. The data source configuration can use variables (for output directories, host names, ports, and so on) so that it can be applied directly across multiple servers unchanged.

Included files can be specified using full paths, paths with variables, or relative paths. Relative paths are resolved by looking in a location relative to the parent file, the server's configuration directory, or the shared configuration directory (in that order).

Include statements can appear anywhere within the `<server.../>` configuration element, and each included file can include other files. To ensure that your included configuration behaves predictably, you must be aware of the following processing rules for included configuration files:

- ▶ Each included file is logically merged into the main configuration at the position that the `<include />` statement occurs in the parent file.

- ▶ For configuration instances such as applications or endpoints, a unique configuration is created for each appearance of the element in the XML file. An ID is generated if not otherwise specified.
- ▶ For singleton services, such as logging or application monitoring, entries are processed in the order that they appear in the file and later entries add to or override previous ones. This is also true for configuration instances (an application or data source) where the configuration instances have the same ID.

Note: Configuration dropins can also be used to further extend the configuration of a server. For more information, see the following link:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.iseries.doc/ae/twlp_setup_dropins.html?cp=SSAW57_8.5.5%2F2-3-11-0-3-1-3-3&lang=en

1.2.8 Dynamic configuration updates

The Liberty profile contains an implementation of the OSGi Configuration Admin service, which operates using the service patterns that are defined in that specification. Per that specification, the Liberty profile configuration manager does the following tasks:

- ▶ Maintains and manages the configuration of the OSGi Service Platform as a whole. It sets the active configuration, at run time, for all configurable services.
- ▶ Monitors the OSGi Service Registry and provides configuration information to configurable services as they are registered.

At start, the configuration manager processes the contents of the `server.xml` file. It then merges those contents with the default values provided by each bundle to provide configuration information to each registered configurable service. If the `server.xml` file changes, the configuration manager reprocesses the file contents and immediately pushes configuration changes out to the configurable services affected by the change. There is no need to restart the server.

The configuration manager also responds dynamically as bundles are added or removed from the system. For example, a new bundle might introduce additional configurable services. When a bundle is started, the configuration manager:

- ▶ Detects new services.
- ▶ Resolves any user-provided configuration against the new configuration metadata and defaults that are provided by the new bundle.
- ▶ Provides the resolved configuration to the consuming services.

Because the run time is dynamic, the configuration manager is tolerant when it processes configuration information from any source (user or system). The configuration manager safely ignores configuration elements or attributes for which it does not have metadata. For unknown attributes, the values are passed to the associated configurable services along with the rest of its configuration data. For unknown configuration definitions, the configuration manager parses configuration information at the time it becomes available and provides it to associated configurable services.

1.3 Runtime composition with features and services

The composable nature of the Liberty profile is based on the concept of features. A feature is a unit of function, essentially the minimum set of bundles that are required to support a particular technology. Features can, and do, overlap, and they can include other features.

As an example, the `servlet-3.0` feature installs the collection of OSGi bundles required to support servlet applications, but not JSP applications. The `jsp-2.2` feature includes the `servlet-3.0` feature (it requires a servlet engine). When you enable the `jsp-2.2` feature in a server, it installs all of the bundles that are required for servlet applications in addition to the bundles required to support JSPs (Figure 1-5).

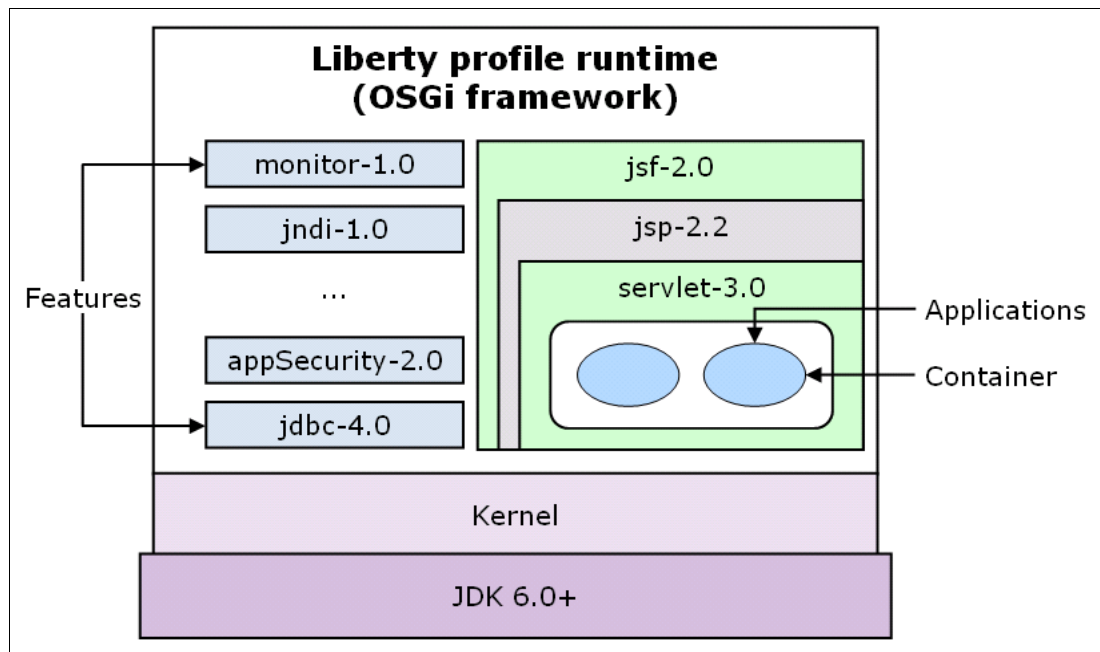


Figure 1-5 Features are the units of composition for a Liberty profile server run time

1.3.1 Feature management

Features are configured in `server.xml` by adding `<feature>` elements to the `<featureManager.. />`. When a new server is created using the `server create` command, the `server.xml` file has the `jsp-2.2` feature enabled, as shown in Example 1-3.

Example 1-3 Server.xml file with jsp-2.2 feature

```
<featureManager>
  <feature>jsp-2.2</feature>
</featureManager>
```

The WebSphere developer tools enhanced editor provides a list of features to choose from and shows those features implicitly enabled by others, as shown in Figure 1-6.

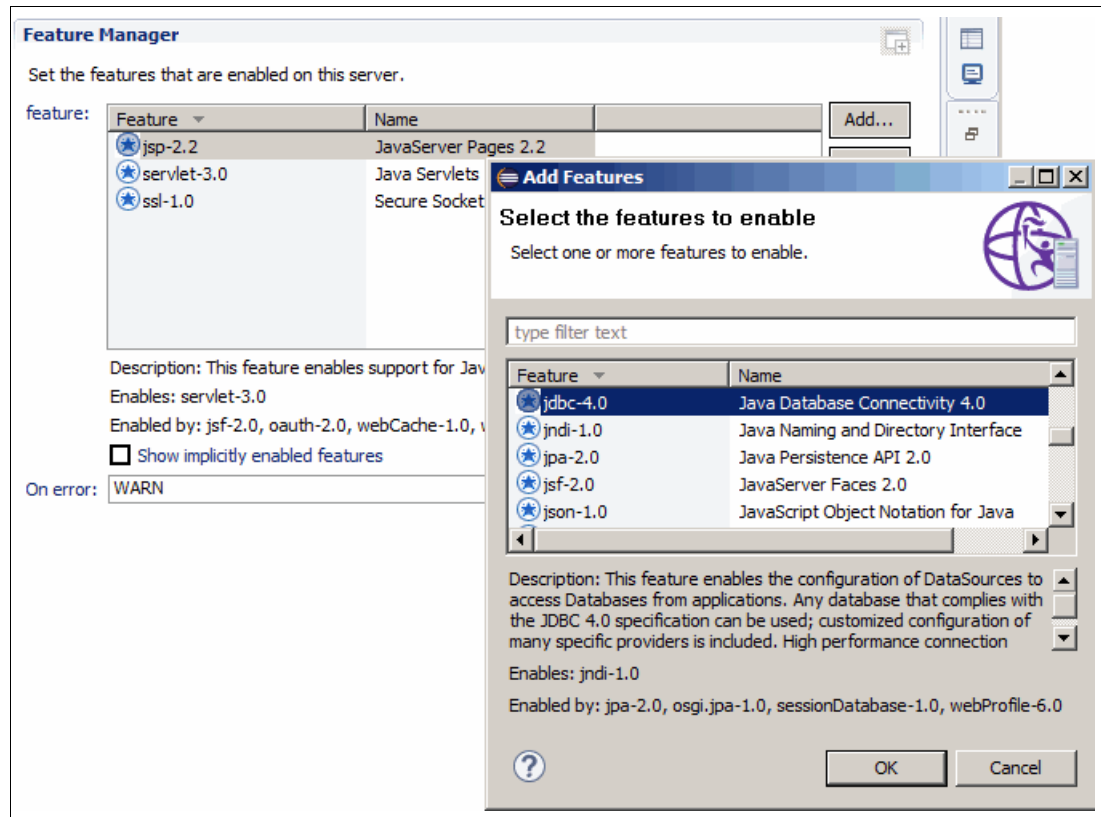


Figure 1-6 Selecting features with the enhanced editor

Features can be added or removed from the server configuration while the server is running without requiring a server restart.

1.3.2 Automatic service discovery

The fit-to-purpose nature of the run time relies on dynamic behavior inherent in the OSGi Framework and the Service Registry. The Liberty profile relies on the OSGi Service Registry for dynamic registration and discovery of services.

When a feature is enabled, the bundles that are required by that feature are installed in the OSGi framework. When a bundle is installed and started, in some cases, services are registered in the Service Registry. For most bundles in the Liberty profile, service registration and lifecycle are managed using OSGi Declarative Services (DS) components.

DS support operates on declared components, each of which is defined by an XML file in the bundle. When a bundle containing component declarations is added to the framework, DS takes action. It reads each component declaration and registers provided services in the service registry. DS then manages the lifecycle of the component by controlling its lifecycle based on a combination of declared attributes and satisfied dependencies.

The simple XML description of components allows DS to resolve service dependencies without requiring the component to be instantiated, or its implementation classes to be loaded. All of this facilitates late and lazy resource loading, which helps improve the server start and runtime memory footprint.

1.4 Frictionless application development

The Liberty profile can deploy applications in the following ways:

- ▶ As an archive file (WAR, EAR, or EBA)
- ▶ Extracted into a directory
- ▶ Described by an XML file (loose configuration)

There is no distinction between installing and starting an application, though installed applications can be stopped and restarted.

By default, the Liberty profile monitors deployed applications for changes. Updates to static files (HTML, CSS, or JavaScript) or JSP files are detected and served immediately. Changes to servlet classes cause an automatic restart of the application.

Caution: For large applications, monitoring the file system for changes to an application can be resource-intensive. Consider extending the monitoring interval if you see excessive I/O activity while the server is running. Application update monitoring should be disabled in production environments. For information, see 8.1.1, “Turning off application monitoring” on page 222.

1.4.1 Quick start using dropins

By default, each server contains a monitored application directory named `dropins`. When an application is placed in this directory, the server automatically deploys and starts the application.

There are several options for placing applications in the `dropins` directory. Each provides a way for the server to determine the application type. The following list describes the placement options:

- ▶ Place the archive file with its identifying suffix (`ear`, `war`, `wab`, and so on) directly into the `dropins` directory:
`${server.config.dir}/dropins/myApp.war`
- ▶ Extract the archive file into a directory that is named with the application name and the identifying suffix:
`${server.config.dir}/dropins/myApp.war/WEB-INF/...`
- ▶ Place the archive file or the extracted archive into a subdirectory that is named with the identifying suffix:
`${server.config.dir}/dropins/war/myApp/WEB-INF/...`

1.4.2 Configuration-based application deployment

Explicitly declaring your application allows you to specify additional properties such as the context root, alter application start behavior, and so on. Explicitly configured applications are recommended for production environments to allow `dropins` monitoring to be disabled.

An application is declared in the `server.xml` file using the `<application.../>` element, which supports the following attributes:

<code>location</code>	The path to the application resource.
<code>id</code>	A unique identifier for the application.

name	A descriptive/display name for the application.
type	The application type: WAR, EAR, or EBA. If this is not specified, it is inferred (if possible) from the application file name.
context-root	The context root of the application.
autoStart	A Boolean value indicating whether the server should start the application automatically. The default is true.

The location is the only required attribute for a declared application. It specifies the location of the application resource (archive, extracted directory, or loose configuration XML file) and can refer to any location in the file system. If the specified location is a relative path, it is resolved first against `${server.config.dir}/apps` and then against `${shared.app.dir}`.

Example 1-4 shows the terms for defining a web application in the `server.xml` file.

Example 1-4 Defining a web application in the server.xml file by using the application element

```
<application context-root="helloworld" type="war" id="helloworld"
  location="helloworld.war" name="helloworld"/>
```

Type-specific application elements can be used to simplify the declaration of applications:

- ▶ `<webApplication.../>`
- ▶ `<osgiApplication.../>`
- ▶ `<enterpriseApplication.../>`

These elements are used in a manner identical to the generic `<application.../>` element, but eliminate the type attribute, as the application type is built into the shorthand of the element itself. The `<webApplication.../>` element also uses a `contextRoot` attribute instead of `context-root`. Example 1-4 and Example 1-5 show two equivalent declarations for an application, using the `<application.../>` and `<webApplication.../>` elements.

Example 1-5 Defining a web application in the server.xml file by using the webApplication element

```
<webApplication contextRoot="helloworld" location="helloworld.war" />
```

Update monitoring for declared applications has a configurable trigger. By default, the file system is polled for changes. When you use the tools to develop your applications, the tools change the update trigger from *polled* to *mbean*. This allows the tools to influence when the server discovers updated files based on when files are saved and whether they are actively being edited.

1.4.3 Using loose configuration for applications

In a loose configuration, an application can be distributed across arbitrary locations on disk, and yet run as though it existed in a single, flat directory structure. Tools such as Rational Application Developer and WebSphere Application Server Developer Tools for Eclipse use loose configuration to allow users to run their applications directly from the projects in their Eclipse workspace. This saves disk space, improves performance, and gives developers more immediate feedback when they make changes because it avoids copying files between the Eclipse workspace and the server definition.

1.4.4 Configuring an application's context root

The context root is the entry point of a web application. The context root for an application is determined using the following criteria in order of precedence:

1. The context root attribute of an application declared in `server.xml`

```
<application context-root="rootOne" location="..."/>, or  
<webApplication contextRoot="rootOne" location="..."/>
```

2. The `context-root` element in the `application.xml` file in an EAR application

```
<context-root>rootTwo</context-root>
```

3. The `context-root` element in the `ibm-web-ext.xml` in a web application

```
<context-root uri="rootThree" />
```

4. The name of the declared web application in `server.xml`

```
<application name="rootFour" ... />
```

5. The `Web-ContextPath` in the bundle manifest of an OSGi WAB

```
Web-ContextPath: /rootFive
```

6. The directory or file name of an artifact in the `dropins` directory

```
${server.config.dir}/dropins/war/rootSix  
${server.config.dir}/dropins/rootSix.war
```

1.4.5 Compatibility with WebSphere Application Server

The Liberty profile is a part of WebSphere Application Server and shares core technologies with the full profile. Applications that are developed on Version v8.5 of the Liberty profile, with the exceptions of those using Java Platform, Enterprise Edition 7, will run on version v8.0 or v8.5 of the full profile. There are differences between the Liberty profile and full profile. The differences are mostly focused on the definition and use of classloading or shared libraries. Section 3.3, "Controlling class visibility in applications" on page 94 provides an overview of how classloading and shared library support work in the Liberty profile. That overview gives special regard to management and use of third-party libraries.

Chapter 8, "From development to production" on page 221 describes strategies for promoting applications that are developed using the Liberty profile to production systems.

1.5 Product extensions

The Liberty profile supports direct extension of the run time by using product extensions. A product extension allows custom content to be added to a Liberty installation in a way that avoids conflicts with the base content of the product and with other product extensions. A product extension is defined by using a simple properties file in the `${wlp.install.dir}/etc/extensions/` directory named `<extensionName>.properties`. This naming convention helps ensure that each product extension has a unique name. The unique name, in turn, is used as a prefix to avoid collisions when specifying extension-provided features and configuration in the `server.xml` file.

Product extensions usually contain features and bundles, but can also contain other things such as custom server templates or scripts. In general, the structure of a product extension resembles the structure of a Liberty profile installation, as shown in Figure 1-7, particularly regarding the following items

- ▶ Bundles (in a `lib/` directory),
- ▶ Features (in a `lib/features/` directory),
- ▶ Server templates (in a `templates/servers/` directory)

Creating a product extension is as simple as creating a properties file in the `${wlp.install.dir}/etc/extensions/` directory, which follows the naming convention and contains the following two values:

- ▶ `com.ibm.websphere.productId`
A descriptive product ID, usually in the form of a reverse-qualified domain name.
- ▶ `com.ibm.websphere.productInstall`
The absolute or relative location of the root product extension directory. If the value of this property is a relative path, it is assumed to be a peer of the Liberty installation directory, `${wlp.install.dir}`.

For convenience in development, the `${wlp.user.dir}/extension/` directory functions as a default product extension, and uses, `usr` as its prefix in the configuration. Figure 1-7 shows the Liberty profile installation with default and product extension content.

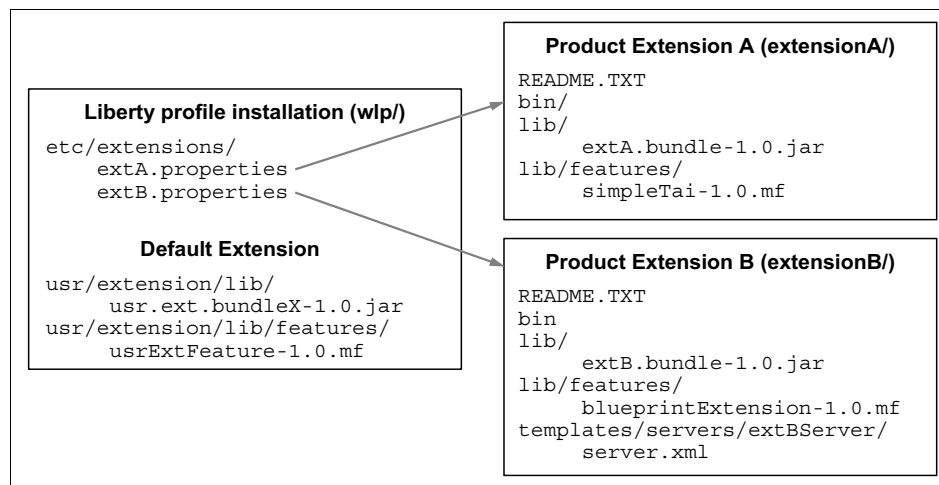


Figure 1-7 A Liberty profile installation with default and product extension content

Example 1-6 provides a snippet of the `server.xml` file demonstrating how prefixes are used when enabling extension-provided features. The prefix that is used for Product Extension A from Figure 1-7 is `extA`, which matches the name of the `extA.properties` file in `${wlp.install.dir}/etc/extensions` that defines the extension.

Example 1-6 Using extension-provided features in the `server.xml` file

```

<featureManager>
  <feature>usr:usrExtFeature-1.0</feature>
  <feature>extA:simpleTai-1.0</feature>
  <feature>extB:blueprintExtension-1.0</feature>
</featureManager>
  
```

Section 9.2, “Defining a custom feature” on page 237 walks through creating and packaging a custom feature as a subsystem archive (an *.esa file). Both the **featureManager** command and the WebSphere developer tools provide mechanisms for managing the features that are installed into product extensions.

For more information about product extensions, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_prod_ext.html

1.6 Overview of Java EE 7

The Liberty Profile has added support for Java EE 7 by means of adding both a bundle for the Java EE 7 Web Profile and a bundle for the Java EE 7 Full Platform. Each of these bundles includes a collection of features that implement the standards required by the Java EE specification.

1.6.1 Java EE 7 Web Profile

The Web Profile provides specifications for web applications. Java EE 6 introduced the Web Profile to help developers of dynamic web applications, providing technologies such as EJB Lite, Java Persistence API, and Java Transaction API. The Java EE 7 Web Profile updates the specifications of the original Web Profile and adds support for HTML5.

The Java EE 7 Web Profile includes the following features:

- | | |
|---------------------------|--|
| beanValidation-1.1 | The Bean Validation 1.1 specification provides an annotation-based model for validating JavaBeans. It can be used to assert and maintain the integrity of data as it travels through an application. |
| cdi-1.2 | The Contexts and Dependency Injection specification makes it easier to integrate Java EE components of different types. It provides a common mechanism to inject component such as EJBs or Managed Beans into other components such as JSPs or other EJBs. |
| ejbLite-3.2 | This feature enables support for Enterprise JavaBeans written to the EJB Lite subset of the EJB 3.2 specification. |
| el-3.0 | This feature enables support for the Expression Language (EL) 3.0. |
| jaxrs-2.0 | This feature enables support for Java API for RESTful Web Services. JAX-RS annotations can be used to define web service clients and endpoints that comply with the REST architectural style. Endpoints are accessed through a common interface that is based on the HTTP standard methods. |
| jaxrsClient-2.0 | This feature enables support for Java Client API for JAX-RS 2.0. |
| jdbc-4.1 | This feature enables the configuration of DataSources to access databases from applications. Any JDBC driver that complies with the JDBC 4.1, 4.0, 3.0, or 2.x specification can be used; customized configuration of many specific providers is included. High performance connection pooling is also provided. |

jndi-1.0	This feature enables the use of Java Naming and Directory Interface (JNDI) to access server configured resources such as DataSources or JMS Connection Factories. It also allows access to Java primitives configured in the server as a jndiEntry.
jpa-2.1	This feature enables support for applications that use application-managed and container-managed JPA written to the Java Persistence API 2.1 specification. The support is built on top of EclipseLink to support the container-managed programming model.
jsf-2.2	This feature enables support for web applications that use the JavaServer Faces (JSF) 2.2 framework. This framework simplifies the construction of user interfaces.
jsonp-1.0	The Java API for JSON Processing (JSON-P) feature provides a standardized method for constructing and manipulating data to be rendered in JavaScript Object Notation (JSON).
jsp-2.3	This feature enables support for JavaServer Pages (JSPs) that are written to the JSP 2.3 specification. This framework simplifies the construction of user interfaces. Enabling this feature also enables the Expression Language (EL) version 3.0 feature.
managedBeans-1.0	This feature enables support for the Managed Beans 1.0 specification. Managed Beans provide a common foundation for different Java EE components types that are managed by a container. Common services provided to Managed Beans include resource injection, lifecycle management, and the use of interceptors.
servlet-3.1	This feature enables support for HTTP servlets written to the Java servlet 3.1 specification. You can package servlets in Java EE specified WAR or EAR files. If servlet security is required, you should also configure an appSecurity feature. Without a security feature, any security constraints for the application are ignored.
webProfile-7.0	This feature combines the Liberty features that support the Java EE 7.0 Web Profile.
websocket-1.0	This feature enables support for WebSocket applications written to the Java API for WebSocket 1.0 specification.
websocket-1.1	This feature enables support for WebSocket applications written to the Java API for WebSocket 1.1 specification.

1.6.2 Java EE 7 Full Platform

Java EE 7 introduces the Full Platform, which includes not just the Web Profile specifications, but also specifications for enterprise, web service, batch, and other applications. It also adds support for application security, deployment, and management.

The Java EE 7 Full Platform bundle adds the following features, but also loads the features from the Java EE 7 Web Profile listed above:

websocket-1.1	This feature enables support for WebSocket applications written to the Java API for WebSocket 1.1 specification.
appClientSupport-1.0	This feature enables the Liberty server to process client modules and support remote client containers.

batch-1.0	This feature enables support for the Java Batch 1.0 API specified by JSR-352.
concurrent-1.0	This feature enables the creation of managed executors that allow applications to submit tasks to run concurrently, with thread context that is managed by the application server. It also enables the creation of managed thread factories to create threads that run with the threadcontext of the component that looks up the managed thread factory.
ejb-3.2	This feature enables support for Enterprise JavaBeans written to the EJB 3.2 specification.
ejbHome-3.2	This feature enables the use of home interfaces in Enterprise JavaBeans.
ejbPersistentTimer-3.2	This feature enables the use of persistent timers in Enterprise JavaBeans.
ejbRemote-3.2	This feature enables the use of remote interfaces in Enterprise JavaBeans.
jacc-1.5	This feature enables support for Java Authorization Contract for Containers (JACC) version 1.5. In order to add the jacc-1.5 feature to your server, you need to add the third-party JACC provider, which is not a part of WebSphere Application Server Liberty profile.
jaspic-1.1	This feature enables support for securing the server runtime environment and applications using Java Authentication SPI for Containers (JASPIC) providers as defined in JSR-196.
javaee-7.0	This feature combines the Liberty features that support the Java EE 7.0 Full Platform.
javaeeClient-7.0	This feature enables support for Java EE 7 Application Client.
javaMail-1.5	This feature allows applications to use the JavaMail 1.5 API.
jca-1.7	This feature enables the configuration of resource adapters to access enterprise information systems (EISs) from applications. The configuration of a resource adapter also includes the configuration of connection factories, administered objects, and activation specifications. Any resource adapter that complies with the JCA 1.7 specification or lower can be used. High performance connection pooling is also provided.
jcaInboundSecurity-1.0	This feature enables security inflow for resource adapters.
jms-2.0	This feature enables the configuration of resource adapters to access messaging systems using the Java Message Service API. This also includes the configuration JMS connection factories, queues, topics, and activation specifications. Any JMS resource adapter that complies with the JCA 1.6 specification can be used.
jmsMdb-3.2	This feature is superseded by mdb-3.2. This feature enables the use of message-driven Enterprise JavaBeans written to the EJB 3.2 specification. Message-driven beans (MDBs) allow asynchronous processing of messages within a Java EE component.
mdb-3.2	This feature enables the use of message-driven Enterprise JavaBeans written to the EJB 3.2 specification. MDBs allow asynchronous processing of messages within a Java EE component.

wasJmsClient-2.0	The wasJmsClient-2.0 feature provides applications with access to the message queues that are hosted on WebSphere Application Server through the JMS 2.0 API. This feature supersedes wasJmsClient-1.1. The wasJmsClient-2.0 feature is compliant with JMS 2.0 specifications and is supported only on JDK 7 or later. It enables access to the messaging engine that is enabled through the wasJmsServer feature, and also to the service integration bus that is hosted on a full profile server.
wasJmsSecurity-1.0	This feature enables the WebSphere Embedded Messaging Server to authenticate and authorize access from JMS clients.
wasJmsServer-1.0	This feature enables an embedded messaging server that is JMS-compliant. Applications can send and receive messages by using the wasJmsClient feature.
wmqJmsClient-2.0	This feature provides applications with access to message queues hosted on IBM MQ through the JMS 2.0 API.



Installation

There are several ways to install the Liberty profile. In this chapter, we focus on the installation methods that are most commonly used for setting up a development environment.

The Liberty profile is designed for easy operation using the provided scripts and direct editing of the server configuration XML files. The development tools provide an integrated development environment (IDE) for application development with plenty of assistance and short-cuts to aid creating, modifying, and controlling Liberty profile servers. The development tools make development easier but are not strictly required, and if you prefer, you can just install the run time.

The chapter contains the following sections:

- ▶ Installing the WebSphere developer tools
- ▶ Installing the Liberty profile
- ▶ Configuring the server runtime JDK
- ▶ Starting and stopping a Liberty profile server

For more installation methods, including installation for production, refer to the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/as_ditamaps/was855_welcome_ndmp.html

2.1 Installing the WebSphere developer tools

WebSphere Application Server Developer Tools for Eclipse is a lightweight set of tools for developing, assembling, and deploying applications to the WebSphere Application Server Liberty profile. These tools should be installed into your Eclipse workbench. Additionally, you can use IBM Rational Application Developer V9, which has WebSphere Developer Tools integrated.

For supported Eclipse versions and more installation methods, refer to the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.0.0/com.ibm.rad.install.doc/topics/t_install_wdt.html

There are a number of different ways to install the WebSphere Application Server Developer Tools for Eclipse into your development environment including:

- ▶ Through the Eclipse Marketplace
- ▶ Using the WASDev site
- ▶ With a downloaded installation file

2.1.1 Installation from Eclipse Marketplace

You can install IBM WebSphere Application Server Liberty Profile Developer Tools for Luna directly from the Eclipse Marketplace by completing the following steps:

1. Start the Eclipse integrated development environment (IDE) for Java EE Developers workbench.
2. Click **Help** → **Eclipse Marketplace** (Figure 2-1).

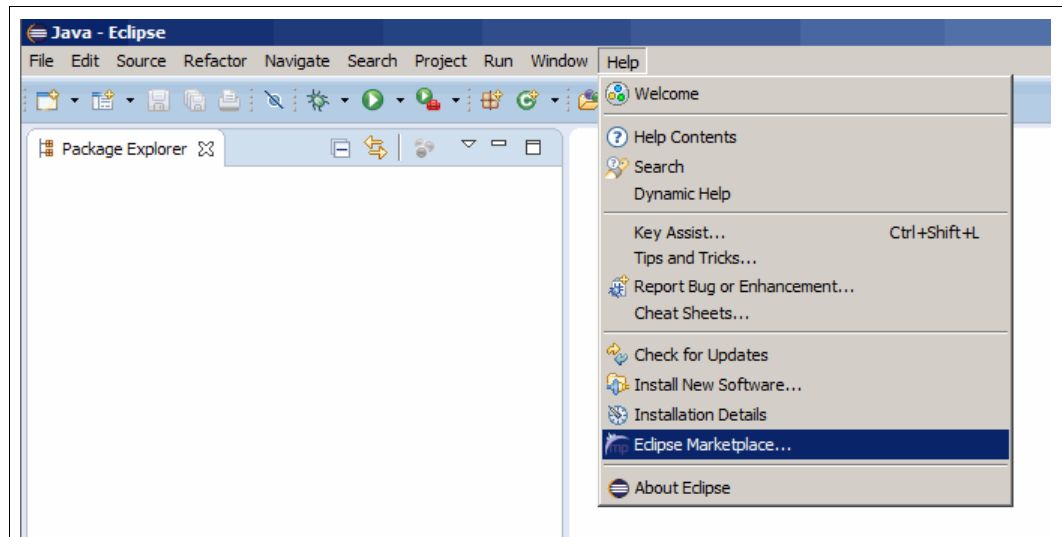


Figure 2-1 Eclipse Marketplace menu item

3. In the Find field, type WebSphere Liberty and then click **Go**.
4. In the list of results, locate **IBM WebSphere Application Server Liberty Profile Developer Tools for Luna** and then click **Install** (Figure 2-2).

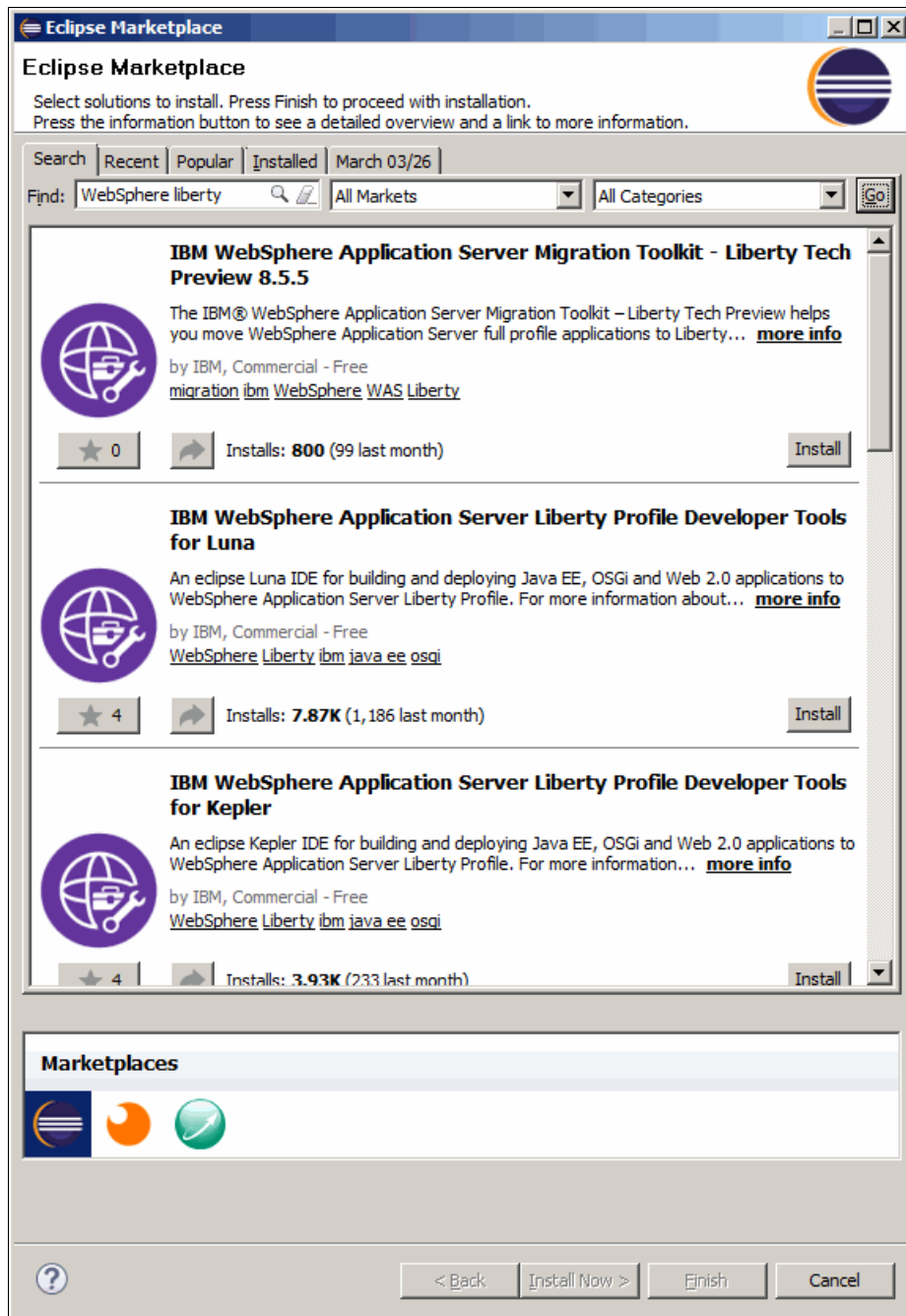


Figure 2-2 Eclipse Marketplace search for WebSphere Liberty

5. On the Confirm Selected Features page, expand the parent nodes and select the features that you want to install. When you are finished, click **Confirm** (Figure 2-3).

The following features are available:

- Open Services Gateway initiative (OSGi) Application Development Tools
- Web Development Tools
- Web Services Development Tools
- WebSphere Application Server Liberty Profile Tools



Figure 2-3 Liberty features

6. On the Review Licenses page, review the license and if you agree to the terms, click **I accept the terms of the license agreement**. Then, click **Finish**. The installation process starts.

Tip: During the installation, a security warning dialog box might open and display the following warning:

“Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the install?”

You can safely ignore the message and click **OK** to continue.

7. When the installation process completes, restart the workbench.

2.1.2 Installation from the WASdev community site

You can install the WebSphere developer tools by using the links that are provided on the WASdev community site if you are using Eclipse 3.7.2 or later. Use the following steps to download the toolset:

1. Start the Eclipse IDE for Java EE Developers workbench.
2. Open a web browser to <http://wasdev.net> and click the **download** tab.

3. Select the appropriate version of the **WebSphere Application Server Developer Tools** from the downloads page (Figure 2-4 on page 41).

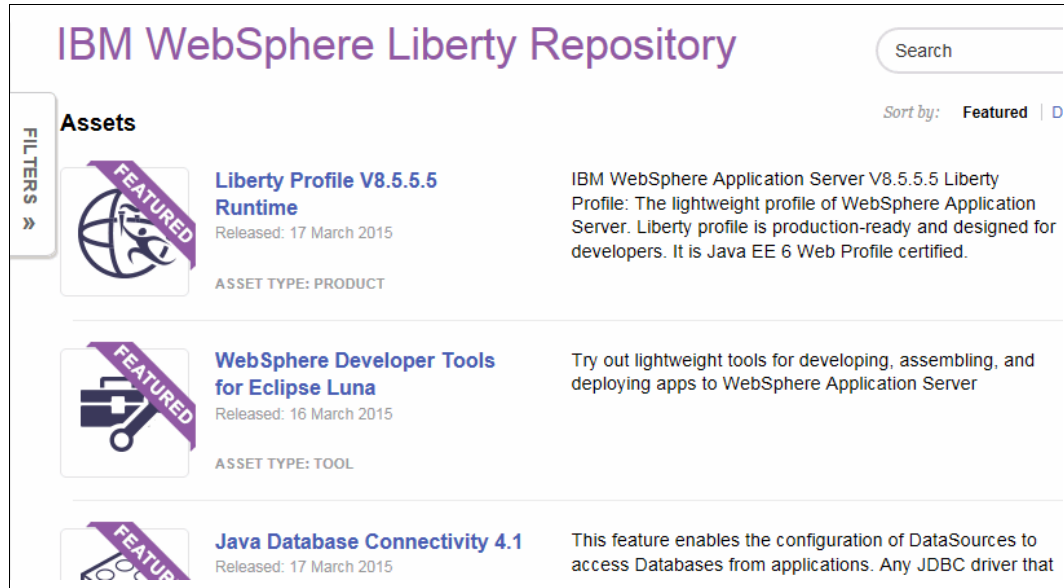


Figure 2-4 IBM WebSphere Liberty Repository

4. Locate **WebSphere Application Server Developer Tools for Eclipse** and click **Download** for your version of Eclipse.
5. Locate the **Install** icon for **WebSphere Application Server Liberty Profile** (Figure 2-5).

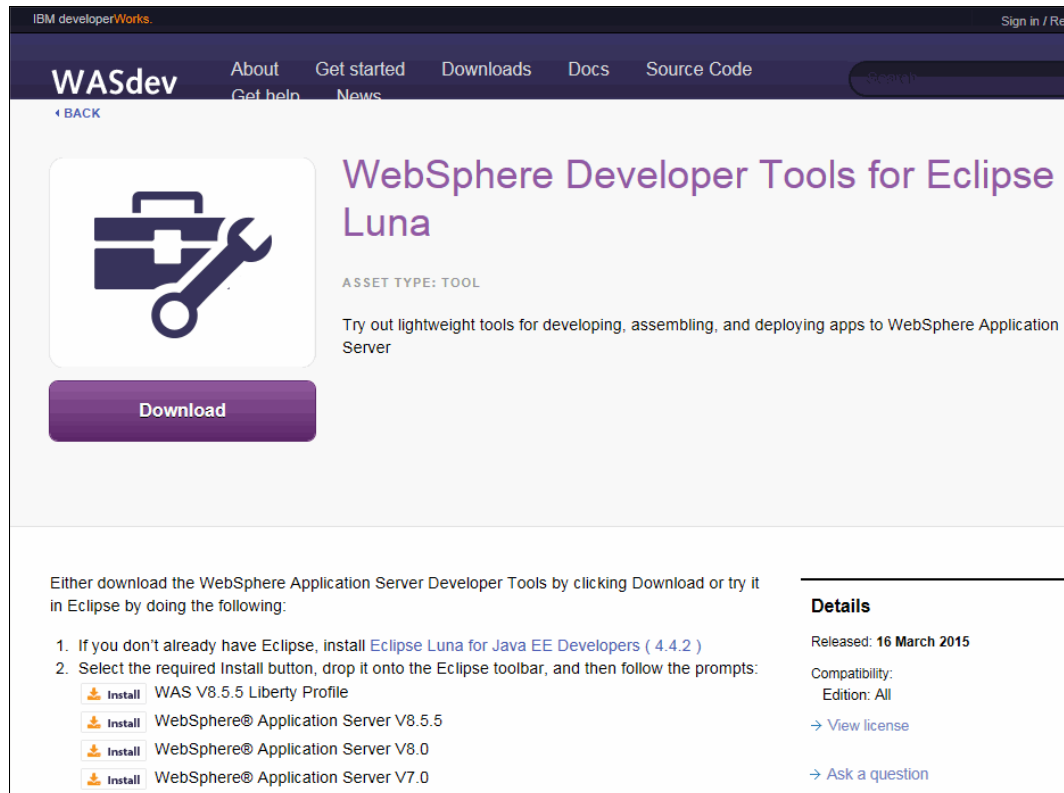


Figure 2-5 WebSphere Developer Tools Install buttons

6. From the browser window, drag the **Install** icon to the toolbar of your Eclipse workbench. This starts the installation process and opens a dialog box.
7. On the **Confirm Selected Features** page, expand the parent node and select the features that you want to install.

The following features are available:

- OSGi Application Development Tools
- Web Development Tools
- Web Services Development Tools
- WebSphere Application Server Liberty Profile Tools

When you are finished, click **Confirm** (Figure 2-6).

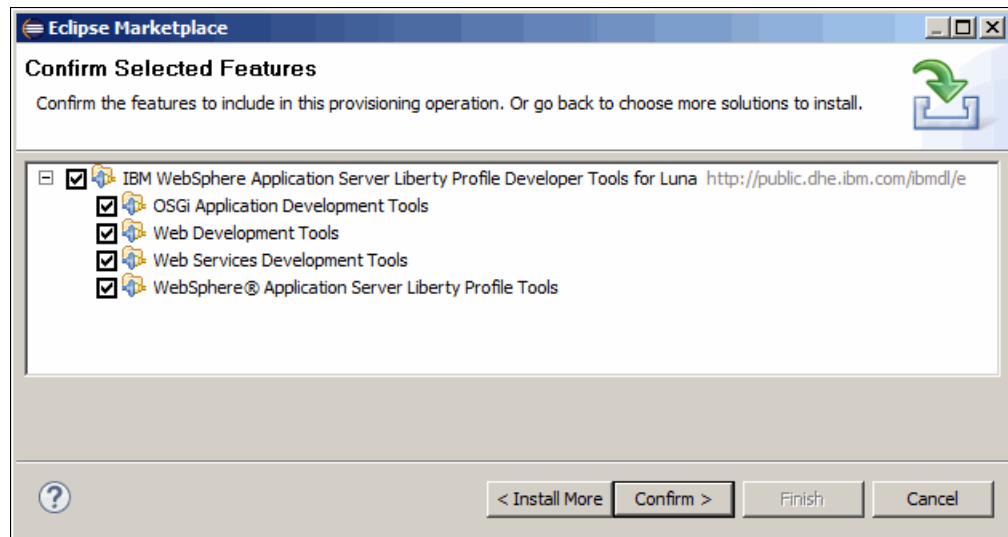


Figure 2-6 Liberty features

8. On the **Review Licenses** page, review the license and if you agree to the terms, click **I accept the terms of the license agreement**. Then, click **Finish**. The installation process starts.

Tip: During the installation, a security warning dialog box might open and display the following warning:

“Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the install?”

You can safely ignore the message and click **OK** to continue.

9. When the installation process completes, restart the workbench.

2.1.3 Installation from downloaded installation files

You can install the IBM WebSphere Application Server Liberty Profile Developer Tools for Luna into an existing Eclipse workbench. This is facilitated by using the installation files that you download to your computer.

If you run the system disconnected from the Internet, Eclipse requires some prerequisite files. For detailed instructions about how to perform this installation and details about the additional prerequisite Eclipse files, refer to the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.0.0/com.ibm.rad.install.doc/topics/t_install_wdt.html

2.2 Installing the Liberty profile

To serve your applications, you must install the Liberty profile runtime environment. The environment is the same whether you are using it for development, test, or production. The WebSphere developer tools integrate with the Liberty profile to give you a seamless, integrated development environment that can aid you in developing and testing your applications.

2.2.1 Installation using the Liberty profile developer tools

The Liberty profile runtime environment can be installed directly from within the Liberty profile developer tools. Installing in this manner also creates an initial server.

Tip: During the configuration of the run time and server, you are prompted to provide names for similar items, which can cause confusion. The following are the names that you will be asked to define, in the order you encounter them:

Server Host Name	The name of the machine running the server or local host.
Server Name	The name of the server as it will be known by Eclipse. This should not be confused with the actual server created in the Liberty profile.
Liberty Profile Server Name	The name of the server that will be created in the Liberty profile runtime environment. This is your actual Liberty Profile Server.

The following steps for installing the Liberty profile and server assume that you have already installed the tools that are described in 2.1, “Installing the WebSphere developer tools” on page 38:

1. Start Eclipse with WebSphere developer tools installed (or Rational Application Developer V9).

2. Select **File** → **New** → **Other**.
3. In the Wizards text box, either browse to **Server** → **Server** or type server in the filter field. Select the **Server** icon and then click **Next** (Figure 2-7).

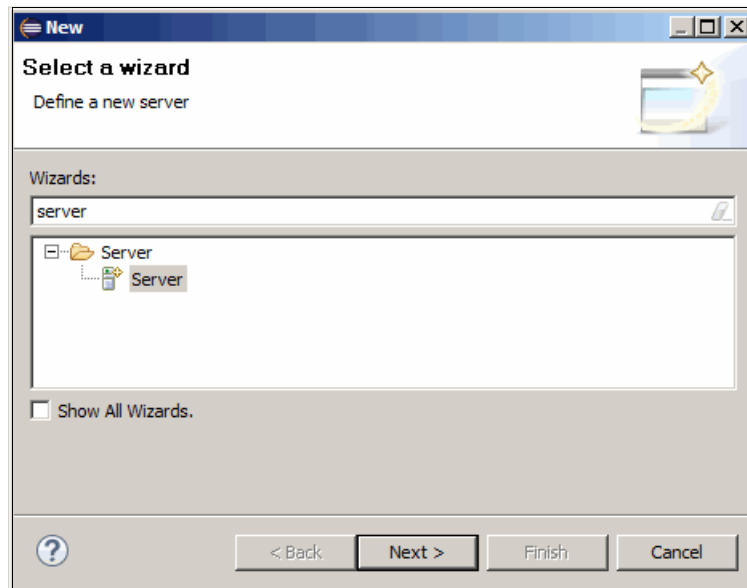


Figure 2-7 Selecting the server installation wizard

- Expand **IBM** and select **WebSphere Application Server Liberty Profile** from the available server list, as shown in Figure 2-8. Provide the *Server's host name* and the *Server name* that you want displayed in the workbench, and click **Next**.

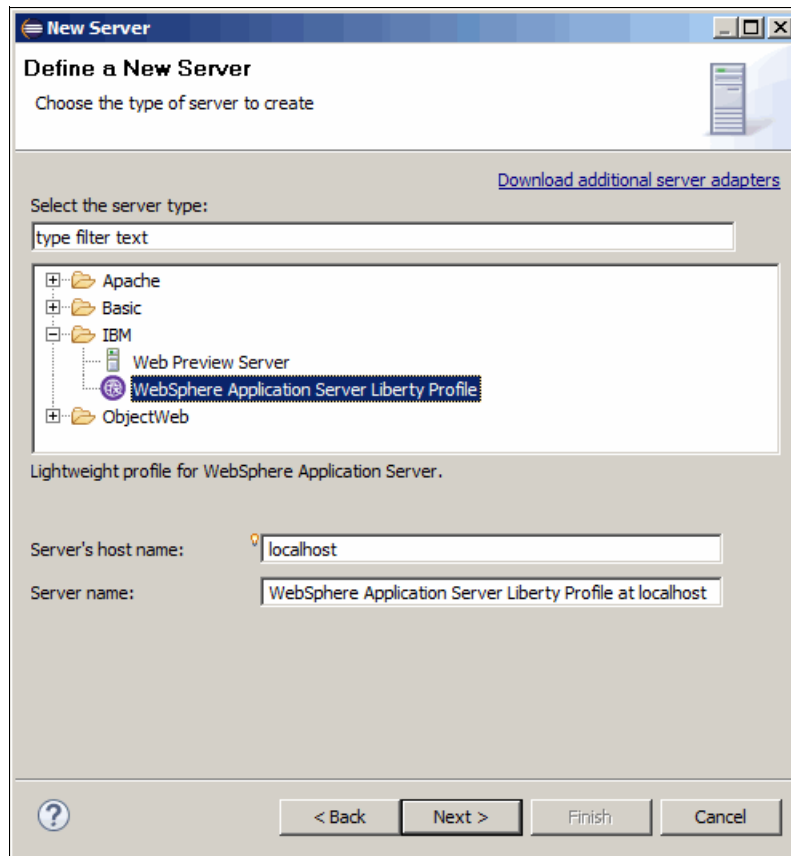


Figure 2-8 Defining the Liberty profile for a new server

5. If you installed a Liberty profile runtime environment, you can select **Choose an existing installation** and browse to select the Liberty profile runtime installation location. This example does not have a Liberty runtime environment that is installed, so one needs to be installed. To do so, select **Install from an archive or a repository** and click **Next** (Figure 2-9).

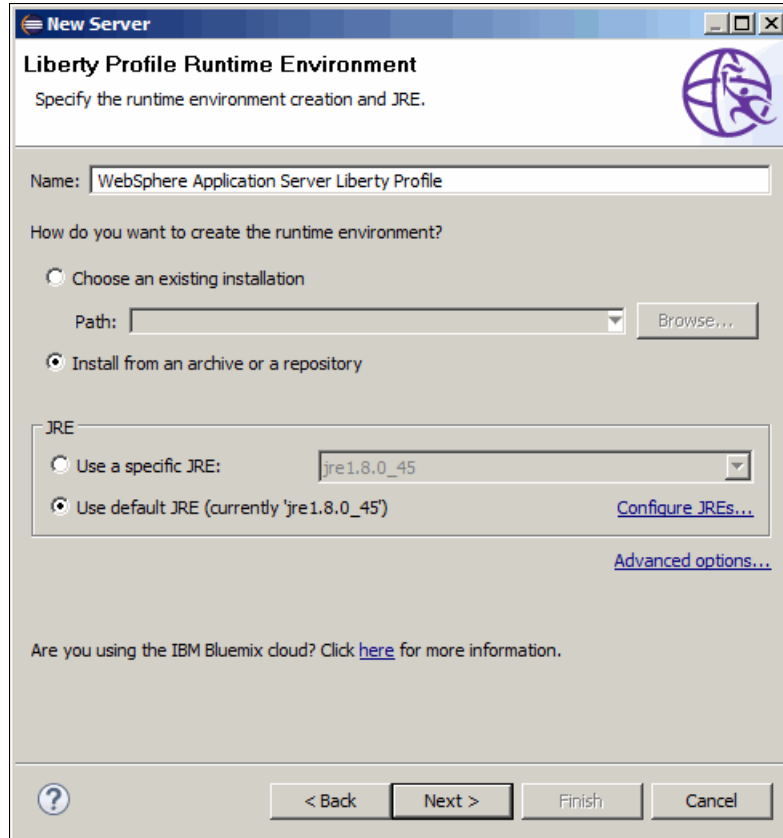


Figure 2-9 Install from an archive or a repository

6. In the **Install Runtime Environment**, start by entering a path for where you would like the runtime installation.
7. If you previously downloaded a Liberty profile runtime archive file from IBM Passport Advantage® or WASdev.net, you can select the downloaded archive using the first option of **Install a new runtime environment from an archive**. In this example, we are downloading directly from the IBM website. Choose the second option **Download and install a new runtime environment from ibm.com**, select the wanted version of **Liberty Profile Runtime**, and click **Next** (Figure 2-10).

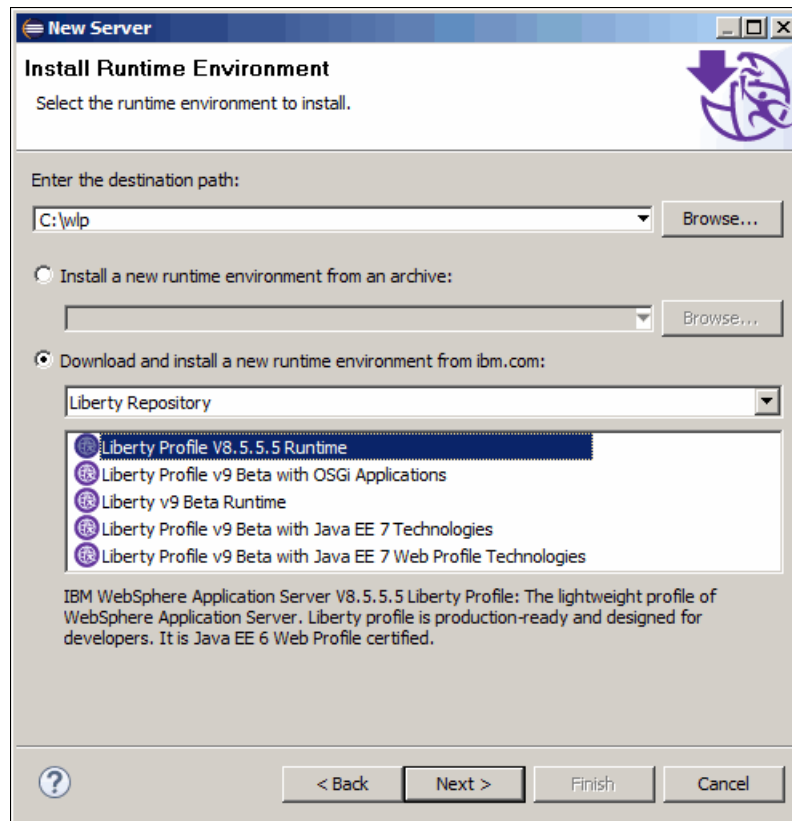


Figure 2-10 Select the Liberty runtime

- On the Install Add-ons page, select any additional features that you want added to your server installation. The additional features include open source packages and the extended version of the Liberty profile, which adds extra capabilities, including JMS and Web Services. Click **Next** (Figure 2-11).

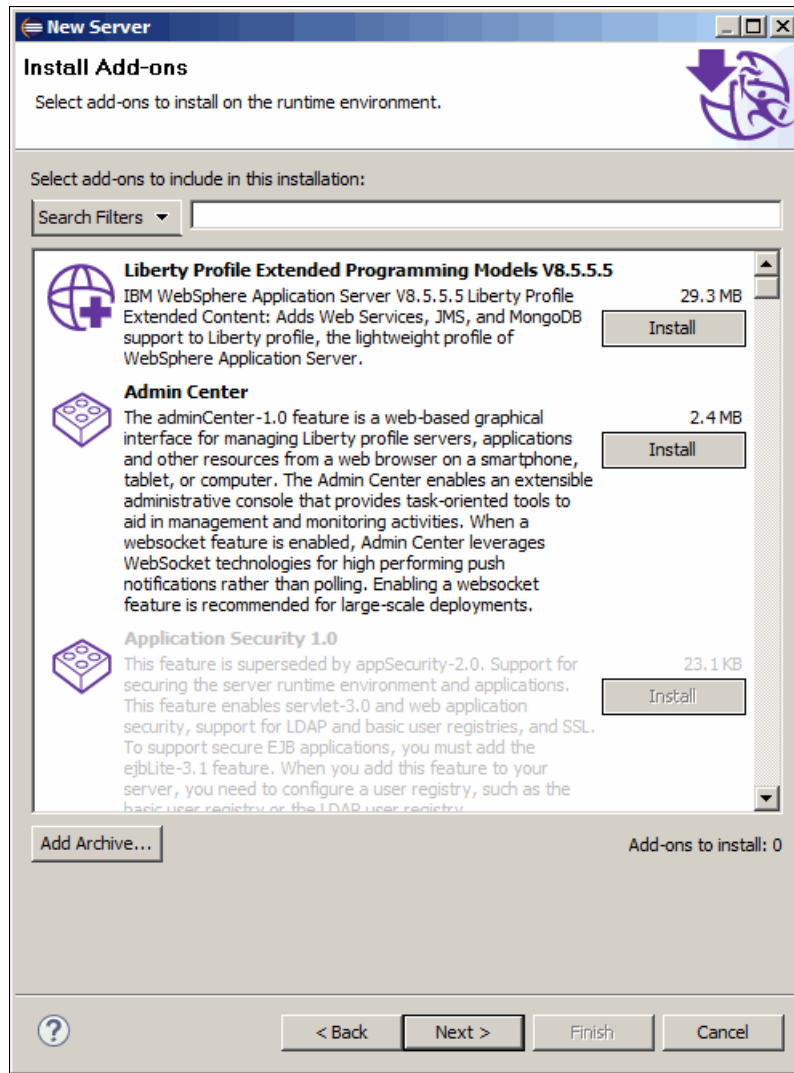


Figure 2-11 Install optional server add-ons

- Accept the license agreements, and then click **Next**.

10. Enter the wanted **Server name** and click **Finish** (Figure 2-12).

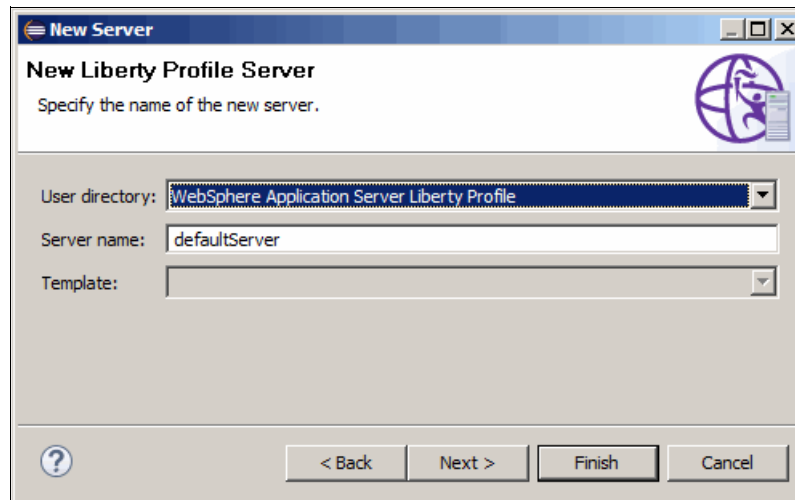


Figure 2-12 Enter the Server name

11. The Liberty profile runtime environment is downloaded and installed.

12. When setup is complete, you see the new server and runtime environment displayed in the Servers view of your workbench. Figure 2-13 shows the newly created server that is expanded.

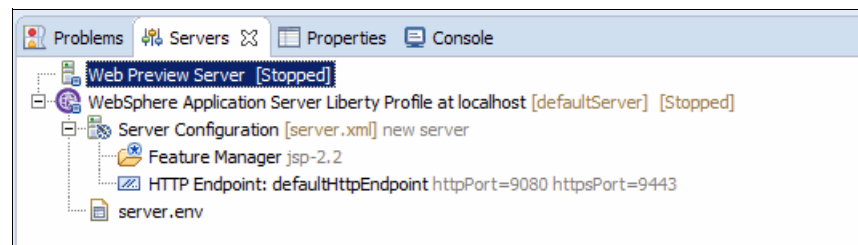


Figure 2-13 The new server shown in the Servers view

Tip: It is also possible to use the WebSphere Developer Tools to do debugging on remote Liberty servers. This requires additional configuration to define where the remote server is running.

2.2.2 Installation using the command line

If you have obtained the Liberty profile runtime environment from WASdev.net or through Passport Advantage, you have downloaded a JAR file. The JAR file has a name similar to wlp-runtime-8.5.5.6.jar.

Installing the runtime environment

To install the Liberty profile, you must extract the archive file by completing the following steps:

1. Run the following command to extract the contents of the Liberty archive:

```
java -jar wlp-runtime-8.5.5.6.jar
```

2. Press x to skip reading the license terms or press Enter to view them.

3. Press Enter to view the license agreement.
4. Press 1 if you agree to the license terms and are ready to proceed.
5. Provide the installation path for the Liberty profile, for example:

C:\IBM\WebSphere

6. Press Enter.

The server is installed into a wlp directory within the install directory that is specified in step 5. If you did not specify an installation location, it installs to the current directory.

Creating a server

The Liberty profile runtime environment does not come with a server defined. To create a server, you must run the following command (Figure 2-14) from the Liberty profile bin directory (wlp/bin):

```
server create <server name>
```

```

Administrator: Command Prompt
C:\IBM\WebSphere>dir
Volume in drive C has no label.
Volume Serial Number is F438-A928

Directory of C:\IBM\WebSphere
05/07/2015  08:46 AM    <DIR>          .
05/07/2015  08:46 AM    <DIR>          ..
05/07/2015  08:51 AM    <DIR>          wlp
               0 File(s)      0 bytes
               3 Dir(s)  101,365,149,696 bytes free

C:\IBM\WebSphere>cd wlp\bin
C:\IBM\WebSphere\wlp\bin>server create liberty-server
Server liberty-server created.

C:\IBM\WebSphere\wlp\bin>

```

Figure 2-14 Create server command

This creates a server, with the provided name, in the usr/servers directory.

Tip: A Liberty profile runtime environment can contain multiple servers. You do not need to install multiple run times onto a machine to run multiple servers.

Server name is an optional parameter. If you do not specify a server name, defaultServer is used.

Adding the new server and runtime environment into the tools

To use the Liberty profile developer tools with your new runtime and server, you must add them into the tools. This can be done by following the instructions in 2.2.1, “Installation using the Liberty profile developer tools” on page 43. Then, select your run time and server at the relevant steps.

Installing using a compressed file

It is also possible to install the runtime using a compressed file. In this example, version 7 of the web profile has been extracted to the local file system using a compressed utility. Other compressed distributions include the javaee7 and osgi bundles.

The example in Figure 2-15 shows the creation of a test server and the feature list for the expanded environment.

```
Administrator: Command Prompt
C:\wlp-webProfile7-8.5.5.6\wlp\bin>server create testserver
Server testserver created.
C:\wlp-webProfile7-8.5.5.6\wlp\bin>productInfo featureInfo
appSecurity-2.0 [1.0.0]
beanValidation-1.1 [1.0.0]
cdi-1.2 [1.0.0]
collectiveMember-1.0 [1.0.0]
distributedMap-1.0 [1.0.0]
ejbLite-3.2 [1.0.0]
el-3.0 [3.0.0]
eventLogging-1.0 [1.0.0]
jaxrs-1.1 [1.0.0]
jaxrs-2.0 [1.0.0]
jaxrsClient-2.0 [1.0.0]
jdbc-4.1 [1.0.0]
jndi-1.0 [1.0.0]
jpa-2.1 [1.0.0]
jsf-2.2 [1.0.0]
json-1.0 [1.0.0]
jsonp-1.0 [1.0.0]
jsp-2.2 [1.0.0]
jsp-2.3 [1.0.0]
ldapRegistry-3.0 [1.0.0]
localConnector-1.0 [1.0.0]
managedBeans-1.0 [1.0.0]
monitor-1.0 [1.0.0]
requestTiming-1.0 [1.0.0]
restConnector-1.0 [1.0.0]
servlet-3.0 [1.0.0]
servlet-3.1 [1.0.0]
ssl-1.0 [1.0.0]
webCache-1.0 [1.0.0]
webProfile-7.0 [7.0.0]
websocket-1.1 [1.0.0]
C:\wlp-webProfile7-8.5.5.6\wlp\bin>
```

Figure 2-15 Web profile feature list

2.2.3 Installation on z/OS

IBM Installation Manager is used to install the Liberty profile on z/OS using the `com.ibm.websphere.liberty.zOS.v85` package offering. Use the latest version of IBM Installation Manager that is supported by your Liberty profile distribution. For more information about installing IBM Installation Manager on z/OS, refer to the following website:

http://www-01.ibm.com/support/knowledgecenter/SSDV2W_1.8.0/com.ibm.silentinstall112.doc/topics/t_zos.html

To install the Liberty profile, install the Liberty offering by running the following `imcl` command of IBM Installation Manager (Example 2-1).

Example 2-1 Installing Liberty profile using IBM Installation Manager on z/OS

```
MTRES1 @ SC49:/u/mtres1/IIM/tools>./imcl install com.ibm.websphere.liberty.zOS.v85
-installationDirectory /u/mtsres1/IBM -repositories
/u/mtres1/kits/8556/repository/ -acceptLicense
Installed com.ibm.websphere.liberty.zOS.v85_8.5.5006.20150425_1310 to the
/u/mtres1/IBM directory.
MTRES1 @ SC49:/u/mtres1/IIM/tools>
```

The following folders are created during the installation process:

- ▶ bin
- ▶ lib
- ▶ lafiles

- ▶ `properties`

No Java Software Development Kit (SDK) is included as part of the Liberty install on z/OS by default. Use a 64-bit Java SDK such as the no-charge feature provided for z/OS, or install one of the optional Java SDKs that are provided along with WebSphere Application Server for z/OS.

Note: On z/OS, create your Liberty server configurations in a location independent of the installed Liberty profile. Do not use the `usr` subdirectory (`wlp/usr`).

2.2.4 The Liberty profile runtime environment and server directory structure

The location of the various files and directories that make up the Liberty profile runtime environment and server are important. You must become familiar with these files to configure and deploy applications to the server.

Runtime directory structure

Figure 2-16 provides an overview of the directory structure that is used by the Liberty profile runtime environment. The directory structure and location of files are essential for the healthy running of your servers.

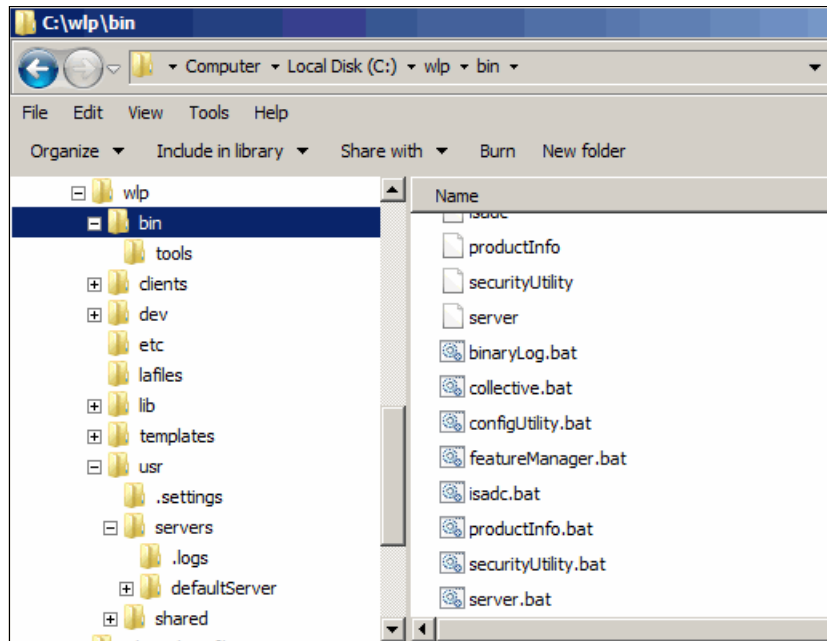


Figure 2-16 The Liberty runtime directory structure

Here are the most important Liberty runtime directories:

- ▶ **bin**
This directory contains scripts that are used to manage the Liberty profile server and server instances.
- ▶ **etc**
This directory is optional. It can be used for customization of the whole Liberty profile installation. The configurations that are defined by files in this directory apply to all Liberty profile servers.
- ▶ **lib**
This directory contains the Liberty runtime libraries.
- ▶ **templates**
This directory contains sample configuration files and a sample `server.xml` file for the Liberty profile.
- ▶ **usr**
This directory contains the server instances with their configuration and applications and any resources that can be shared between servers. The `usr` directory is user-owned and as such is not modified by any service releases.

Server directory structure

Similar to the runtime directory structure, it is essential that you become familiar with the location and layout of the Liberty profile server files and directories (Figure 2-17).

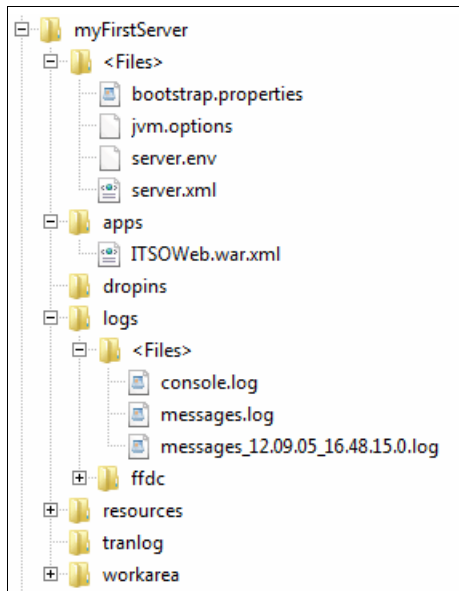


Figure 2-17 The Liberty server directory structure

Here are the most important Liberty profile server directories:

- ▶ apps
This directory is optional. It can contain deployed applications or application descriptors if they are deployed using the WebSphere developer tools.
- ▶ dropins
Applications can be added and removed to the liberty server simply by dropping them into this folder. This directory is constantly monitored by the server to identify any changes and add or remove an application when the changes occur.
- ▶ logs
Contains the logs that are produced by the Liberty profile server. By default, this is the place where the trace or message logs are written. It also contains the first-failure data captures (FFDCs).
- ▶ resources
Contains additional resources for the Liberty profile server instance. For example, keystores that are generated by the Liberty profile server are in this directory.
- ▶ tranlog
Contains the transactional logs that are produced by the server run time and the applications. The transactional logs are used to commit or roll back transactional resources if there is a server failure.
- ▶ workarea
Contains the Liberty profile server operational files; is created during the first server run.

2.3 Configuring the server runtime JDK

The Liberty profile requires a Java Runtime Environment (JRE) to run in. The Liberty profile server runs on Java version 6 or above provided by any vendor. A full list of supported Java versions is noted in the following System Requirements document:

<http://www-01.ibm.com/support/docview.wss?uid=swg27038218>

The following sections describe the ways that you can define which JRE is used.

2.3.1 Defining the JRE from within workbench

When running a server through the workbench, the JRE to be used is defined in the server properties. When you created the server in Eclipse, you could define the JRE. To change the JRE, complete the following steps:

1. Switch to the Servers view in the workbench.
2. Right-click your server and select **Open** (Figure 2-18).

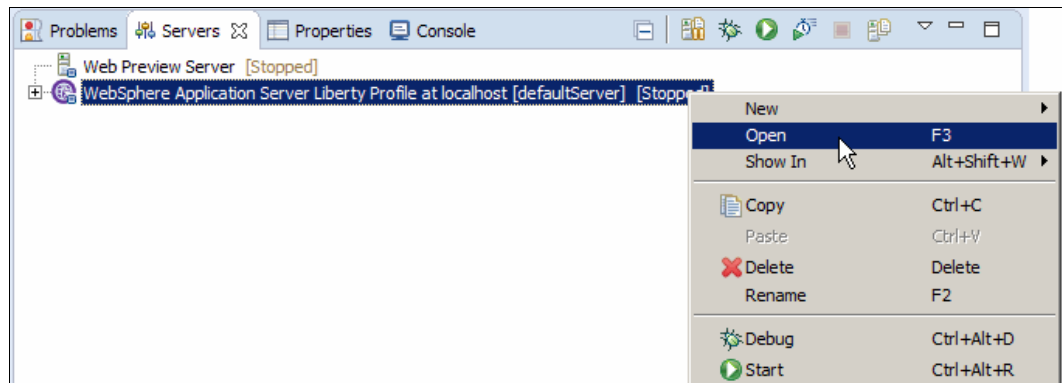


Figure 2-18 Opening a Server Properties window

3. Select the **Runtime Environment** link (Figure 2-19).

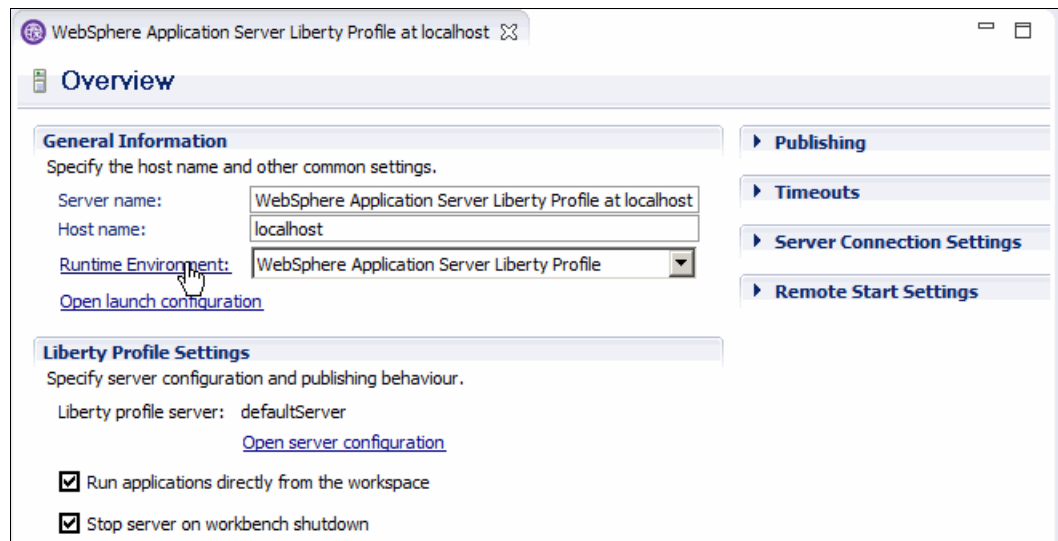


Figure 2-19 Opening the Runtime Environment properties

4. A window is displayed allowing you to change the runtime environment, including changing your JRE (Figure 2-20). After you are satisfied with your changes, click **Finish**.

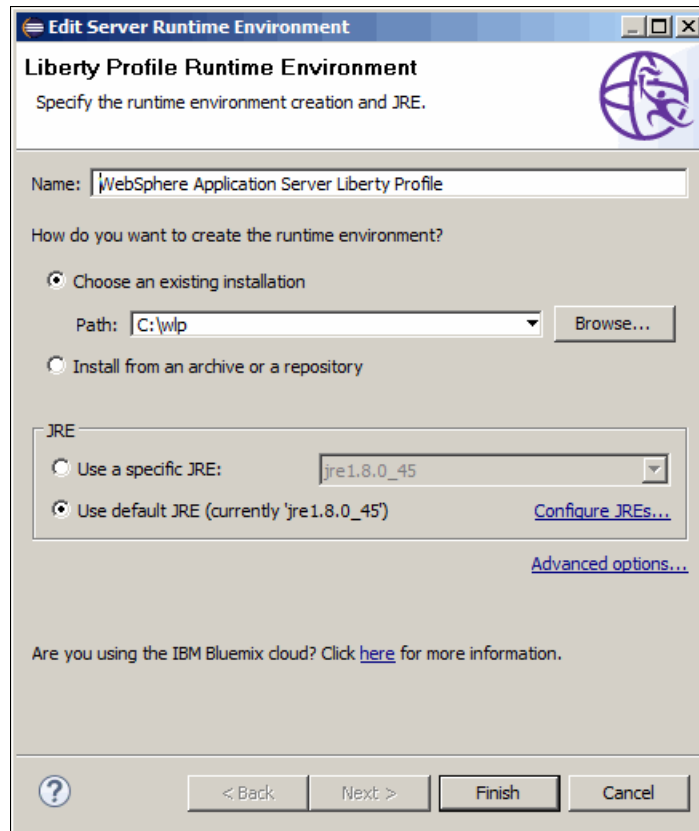


Figure 2-20 Edit Server Runtime Environment window

5. Save the changes to the Server properties. The new settings are used when you next start the server.

Caution: If you created a server .env file that defines the JRE (either for the server or for the run time), this takes preference over the Eclipse setting. You must remove this file in order for the Eclipse setting to work.

2.3.2 Configuring the system JRE

When starting the Liberty server using the command line, the server must know where to find its JRE. In the absence of any server or runtime configuration files that define the JRE, the system JRE is used.

On Windows systems, you can use Example 2-2 to set the JAVA_HOME property, so the Liberty server uses your own JRE installation directory.

Example 2-2 Setting the JRE for Liberty on Windows

```
set JAVA_HOME=C:\IBM\jre6
set PATH=%JAVA_HOME%\bin;%PATH%
```

On Linux Systems, you can use Example 2-3 to set the JAVA_HOME property.

Example 2-3 Setting the JRE for Liberty on Linux

```
export JAVA_HOME=/opt/IBM/jre6
export PATH=$JAVA_HOME:$PATH
```

The Liberty profile runtime environment searches for the Java command in the following order of properties: JAVA_HOME, JRE_HOME, and PATH.

2.3.3 Using server.env to define the JRE

The server.env file is a Liberty profile-specific configuration file that can be used to define the JRE for the servers to use. When you install the Liberty profile this configuration file does not exist, so you must create it. Create the file in one of the following locations:

- ▶ `${wlp.install.dir}/etc/server.env`

The settings here apply to all servers in the runtime environment.

- ▶ `${server.config.dir}/server.env`

These settings are specific to the server containing the file. This server.env is used in preference to the runtime level when both exist.

To define the JRE, add the following line into the server.env file:

```
JAVA_HOME=<path to your JRE>
```

You can use the server.env file to provide any other additional environment variables that your server might need.

Caution: The server.env files support only key=value pairs. Variable expansion is not supported.

2.3.4 Specifying JVM properties

You can use the jvm.options files at the runtime and server levels to specify jvm-specific start options, for example, -X arguments. The options are applied when you start, run, or debug the server. Be sure to specify only one option per line.

When you install the Liberty profile, this configuration file does not exist, so you must create it. Create the file in one of the following locations:

- ▶ `${wlp.install.dir}/etc/jvm.options`

The settings here apply to all servers in the run time.

- ▶ `${server.config.dir}/jvm.options`

These settings are specific to the server containing the file. This jvm.options file is used in preference to one at the runtime level when both exist.

2.4 Starting and stopping a Liberty profile server

You can control the state of the Liberty profile server either from the command line or through the workbench. We describe the two methods in this section.

2.4.1 Starting and stopping the server by using the command line

The server can be controlled from a single **server** command that is in the `${wlp.install.dir}/bin` directory. The following commands can be used to start and stop the server:

- ▶ `server start <servername>`
Starts the server running in the background
- ▶ `server run <servername>`
Starts the server running in the foreground and writes the console output to the window. To stop the Liberty profile server in this mode, press Ctrl+c or kill its process or run `server stop` from another command window.
- ▶ `server debug <servername>`
Starts the server in debug mode.
- ▶ `server stop <servername>`
Stops the server.
- ▶ `server status <servername>`
Displays the current state of the server.

Hint: If the workbench is open and it has the Liberty profile server already defined, it automatically updates. Updates reflect the status of the server and show the output from the server in the console window.

2.4.2 Starting and stopping the server from the workbench

You can control the state of the Liberty profile server from directly within the workbench. This can be done from the Server view. Either right-click the server and select the required option, or click the icons that are attached to the Server view, as shown in Figure 2-21.

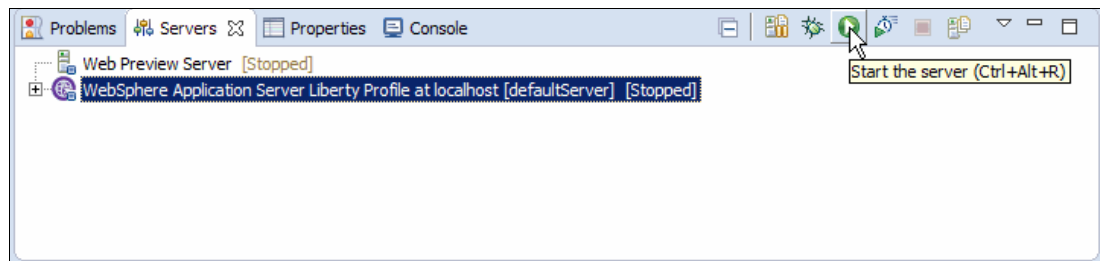





Figure 2-21 Starting the server by using the start button

The following buttons are available to control the server:

-  Starts the server in debug mode.
-  Starts the server.
-  Stops the server.



Developing and deploying web applications

IBM WebSphere Application Server V8.5 Liberty Profile provides a simplified environment for developing and deploying applications. Whether working in the Liberty profile developer tools or in a third-party text editor, you can quickly build, run, and update applications. Development impact, such as configuration and server restarts, are kept to a minimum to reduce the amount of time it takes to develop new applications. The Liberty profile supports Java SE 6, Java SE 7, and Java SE 8 runtime environment, but the Java EE 7 features require Java SE 7 or later. In this chapter, we describe the benefits of developing applications in the Liberty profile environment.

The chapter contains the following sections:

- ▶ Developing applications using the Liberty profile developer tools
- ▶ Developing outside the Liberty profile developer tools
- ▶ Controlling class visibility in applications

3.1 Developing applications using the Liberty profile developer tools

The Liberty profile developer tools provide a lightweight, rich environment for developing applications. Using the Liberty profile developer tools simplifies the creation of applications by providing wizards and graphical design tools for Java EE applications. Server configuration tasks are simplified by using the tools. In many cases, the tools can infer what configuration is needed based on the application and update it automatically. Working in the Liberty profile developer tools environment also enables iterative development by republishing applications when you make changes in the integrated development environment (IDE).

In this section, we describe an iterative development scenario involving several technologies that are supported by the Liberty profile server.

3.1.1 Using the tools to create a simple servlet application

In this section, we describe how to develop and deploy a simple servlet application using either the Liberty profile developer tools or a simple text editor. Following the procedure that is described in this chapter reveals the simplicity of developing and updating applications in the Liberty environment.

Liberty profile supports the Java EE Servlet 3.1 API. This book assumes some knowledge of Java EE technologies, so the actual coding of the example servlet is not described here.

The Liberty profile developer tools provide a rich development environment for building web applications. In this section, we take you through the end-to-end process of developing, deploying, and updating an application.

Getting started

Before you begin, make sure that you have installed the Liberty profile server and tools as described in 2.1, “Installing the WebSphere developer tools” on page 38. The following steps outline how to create a new project for later deployment:

1. Begin by creating a web project in the Liberty profile developer tools by using the Create a Web Project wizard from the toolbar, as shown in Figure 3-1.

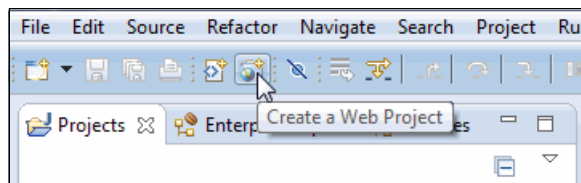


Figure 3-1 Selecting the Create a Web Project wizard

2. In the first window, give the project the name ITS0Web and select the **Simple** project template. Leave Programming Model, as the default value, Java EE. Clicking **Next** takes you to the Deployment options.
3. Under Target Runtime, make sure that **WebSphere Application Server V8.5 Liberty Profile** is selected. Leave all other options as the default values, and click **Finish**. Accept the option to switch to the web perspective if you are prompted to do so.

Building a servlet application

Now add a servlet to the ITSOWeb web project by completing the following steps:

1. Begin by creating a servlet using the Create Servlet wizard:
 - a. To start the wizard, right-click your **ITSOWeb** project in the Enterprise Explorer window and select **New** → **Servlet**.
 - b. Use `com.itso.example` for the package name and `HelloITSO` for the servlet class name. All other values can be left at the defaults. Figure 3-2 shows the Create Servlet wizard.

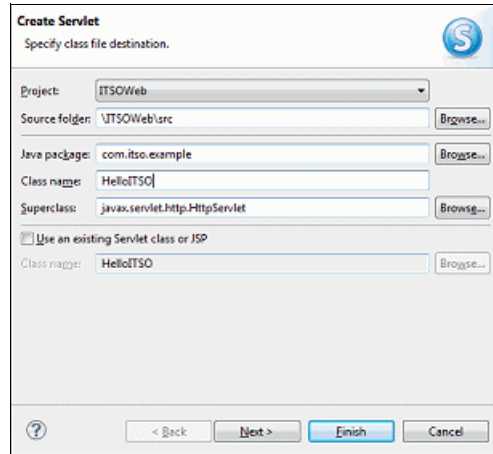


Figure 3-2 Create Servlet wizard

2. After finishing the wizard, the `HelloITSO.java` file is included in the ITSOWeb project. Open the file and add some simple output to the servlet's **doGet** method. Figure 3-3 shows an example **doGet** method.

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    PrintWriter pw = response.getWriter();
    pw.println("<BODY>");
    pw.println("<H2>Hello, Liberty developer</H2>");
    pw.println("</BODY>");
}
```

Figure 3-3 Example doGet method for the HelloITSO servlet

For more information about developing servlets and other dynamic web content, see the Liberty profile developer tools at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.0.0/com.ibm.etools.rpe.doc/topics/crpe.html

Deploying the application

Before deploying the application, ensure that you created a Liberty Profile Server in the development environment as described in 2.2.1, “Installation using the Liberty profile developer tools” on page 43. To deploy the application, complete the following steps:

1. In the Servers view, locate your Liberty Profile Server. Right-click the server and select **Add and Remove**.

2. In the resulting window, select the **ITS0Web** project and click **Add**, then click **Finish**. The web project is now available on the server. You should see output similar to Figure 3-4 on page 62 in the console. If your server is not started, start your server. The application is automatically assigned the context root matching the project name of ITS0Web. The application start is fast.

```
[AUDIT ] CWWKG0016I: Starting server configuration update.  
[AUDIT ] CWWKT0016I: Web application available (default_host): http://localhost:9080/ITS0Web/*  
[AUDIT ] CWWKZ0001I: Application ITS0Web started in 0.8 seconds.  
[AUDIT ] CWWKG0017I: The server configuration was successfully updated in 0.09 seconds.
```

Figure 3-4 Output from deploying an application

Result: Note the extreme speed of the application start and that it was automatically assigned the context root matching the project name of ITS0Web.

Verifying the application

To verify the application, open a web browser and go to:

<http://localhost:9080/ITS0Web>HelloITS0>

You should see the following output syntax:

Hello, Liberty developer

Updating the application

The Liberty profile server supports hot deployment for all applications. To verify, change the servlet to output a different message in its **doGet** method. When you save the file, the changes are automatically published to the server. In the console output, notice that the application was stopped, updated, and made available again almost instantaneously. Verify that your changes are active by visiting the HelloITS0 page again.

3.1.2 Developing and deploying a JSP application

The Liberty profile server supports the JavaServer Pages (JSP) 2.3 specification. JSP allows you to separate presentation logic from your business logic.

The procedure for creating and deploying a JSP in the Liberty profile developer tools is similar to the procedure for a servlet. To create and deploy, complete the following steps:

1. Begin by creating a new JSP file by right-clicking the **ITS0Web** project and selecting **New** → **JSP File**.
2. Enter `ITS0Welcome.jsp` for the name of the page and click **Next**.
3. On the Select JSP template window, make sure that the template **New JSP File (HTML)** is selected and click **Finish**.

4. The `ITSOWelcome.jsp` page automatically opens in the Rich Page editor. By default, the editor displays a split view that shows both the JSP source and a preview of the output. Insert some text into the source editor between the body tags. Notice that the preview pane is automatically updated. Figure 3-5 on page 63 shows the JSP editor after adding a form to submit a simple input value to the `HelloITSO` servlet.

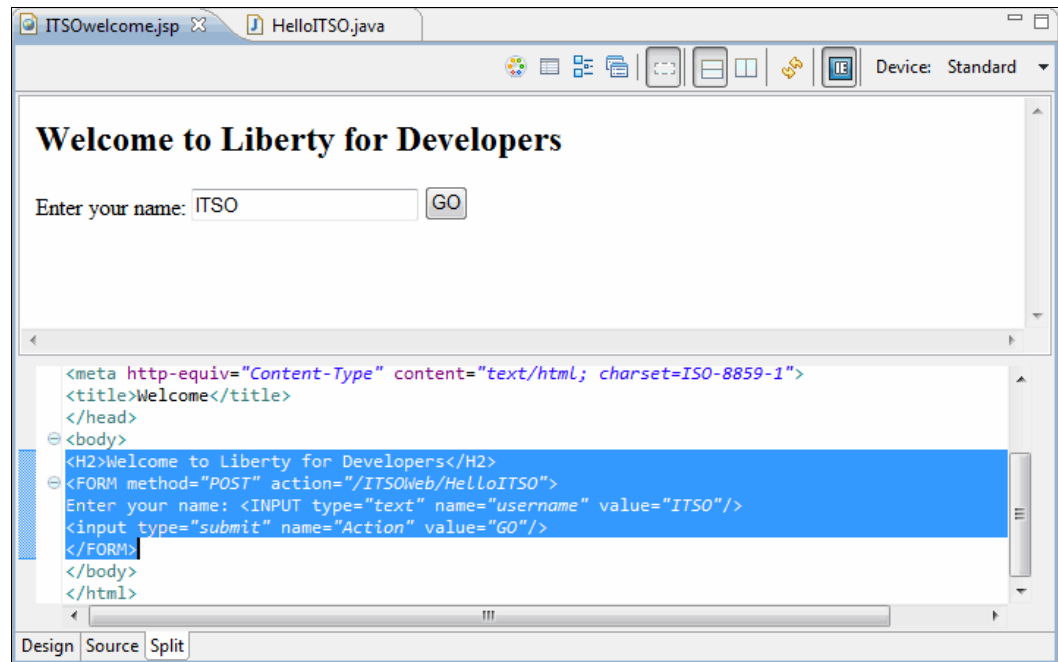


Figure 3-5 Editing the `ITSOWelcome` JSP file

The Rich Page editor contains several powerful features that are beyond the scope of this book. For more information, refer to the following website:

http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.0.0/com.ibm.etools.rpe.doc/topics/crpe.html

5. Because the ITS0Web project is already deployed to the ITS0 Example Server, no extra steps are needed to deploy and test the new JSP file. However, to test the JSP file, which is shown in Figure 3-6, open the **HelloITS0** servlet again and add an implementation for the **doPost** method. For example, the code, which is shown in Figure 3-6, outputs a message using the value of the JSP's input field. When you are finished editing the servlet, save and close the file. Notice that the application is automatically updated on the server.

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    hello(response.getWriter(), request.getParameter("username"));
}

private void hello(PrintWriter writer, String name) {
    writer.println("<BODY>");
    writer.println("<H2>Hello, " + name + "</H2>");
    writer.println("</BODY>");
}
}
```

Figure 3-6 Example `doPost` implementation for the `HelloITS0` servlet

6. To test the application, enter the link in to your web browser:

`http://localhost:9080/ITS0Web/ITS0Welcome.jsp`

You should be able to submit a value in the JSP file and see the output message from the servlet that reflects the value that is given in the input field of the JSP.

3.1.3 Developing and deploying a JSF application

JavaServer Faces (JSF) simplifies the development of user interfaces for web applications by providing the following features:

- ▶ Templates to define layout
- ▶ Composite components that turn a page into a JSF UI component
- ▶ Custom logic tags
- ▶ Expression functions and validation
- ▶ Component libraries
- ▶ XHTML page development

For more information about JSF and the features it provides, refer to the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.mutlplatform.doc/ae/cweb_javaserver_faces.html

JSF tools

The Liberty profile developer tools provide a rich development environment for building high-quality JSF applications. In this section, we describe a simple JSF application using JSP files, a managed bean, and navigation rules. For more information about the powerful features that are provided by the JSF tools, refer to the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.0.0/com.ibm.etools.jsf.doc/topics/tjsfover.html

Enabling JSF in a web project

To enable JSF in a web project, right-click the project and select **Properties**. Click **Project Facets** and enable **JavaServer Faces**, as shown in Figure 3-7. Click **OK** to close the properties.

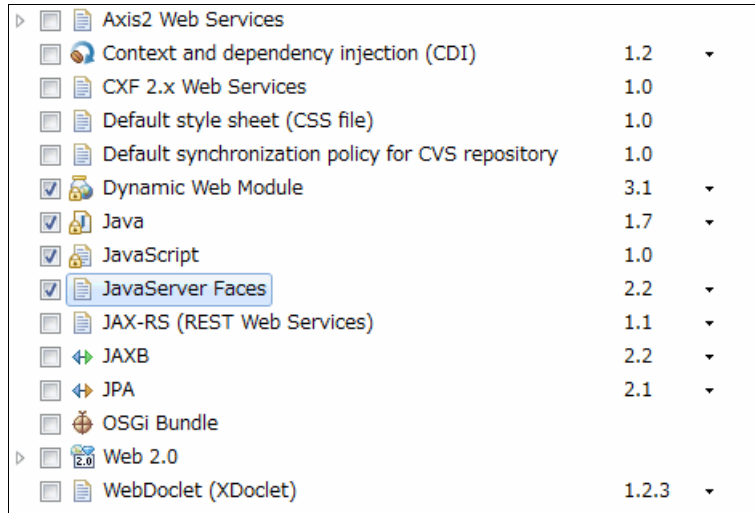


Figure 3-7 Enabling JSF in a web project

Feature enablement

To reduce the impact and resource usage, the Liberty profile server enables only features that are actively being used. When you create a server using the Liberty profile developer tools, the configuration automatically is updated to enable servlets and JSPs. By default, the JSF application run time is not enabled. The server configuration must be updated to run JSF applications.

When you add applications to the server or update the facets of a project that is deployed to the server, the Liberty profile developer tools try to determine what features should be enabled on the server to support the application. If you update the project to add the JSF facet, the tools prompt you to resolve conflicts to add the jsf-2.2 feature to the Liberty profile server. You can add or remove the features by **Add** or **Remove** and resolve conflicts, as shown in Figure 3-8 on page 66.

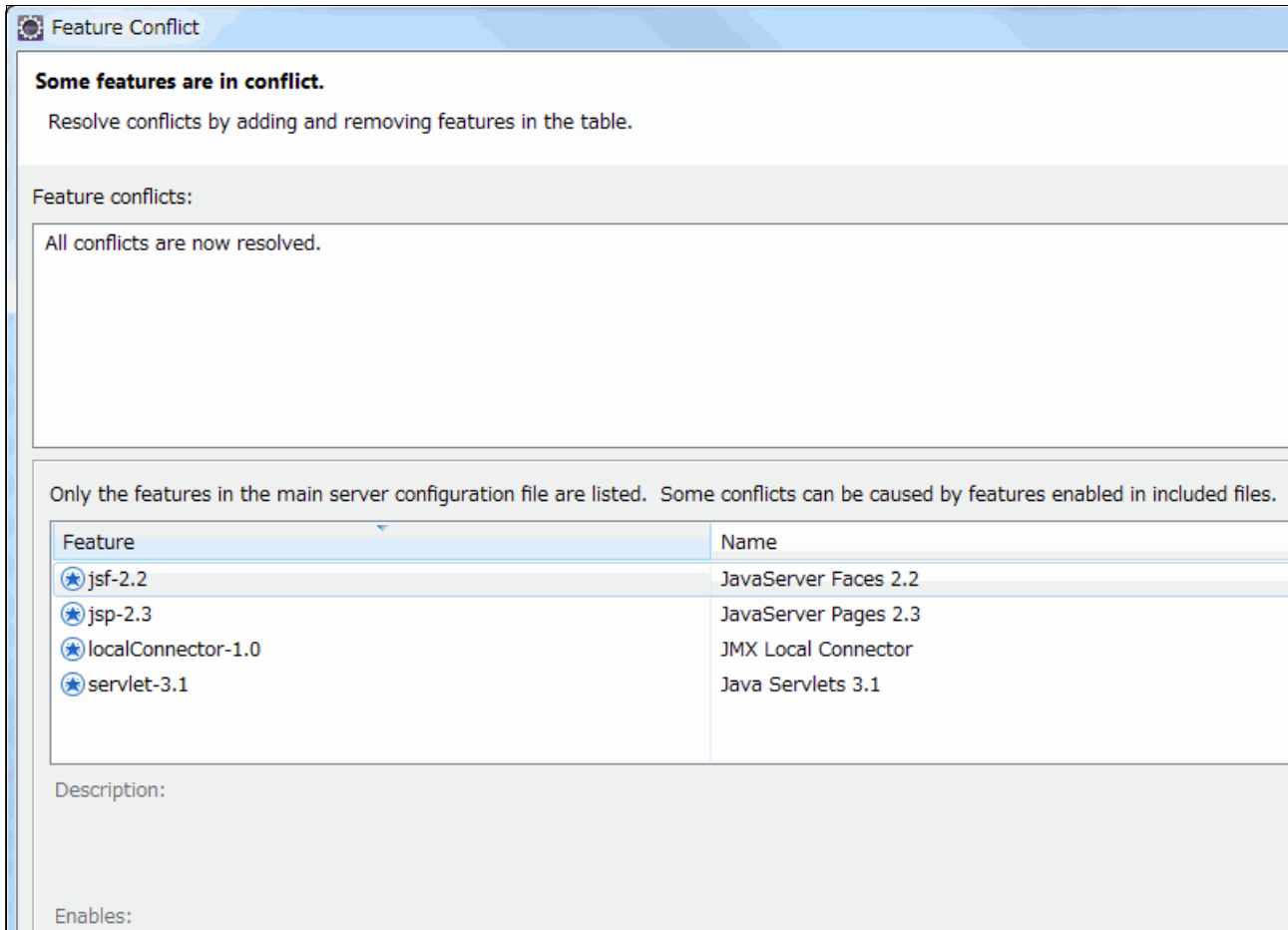


Figure 3-8 JSF feature enablement

You can verify that the change took place by opening the server.xml file and confirming that `<feature>jsf-2.2</feature>` was added. Also, the console output shows that the jsf-2.2 feature was installed to the server, as shown in Figure 3-9.

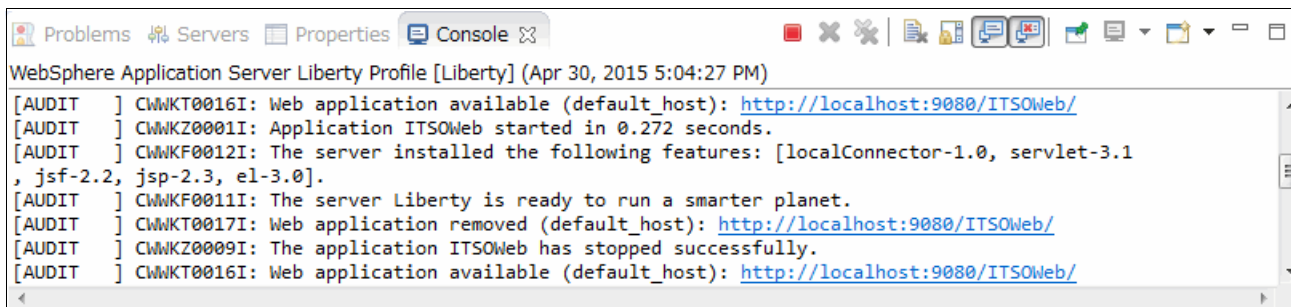


Figure 3-9 Console output after JSF feature enablement

Building a JSF application

Now that JSF is enabled in your web project, begin creating a JSF application by opening the `WebContent/WEB-INF/faces-config.xml` file for editing in the Faces Config editor.

Adding a managed bean and set properties

Complete the following steps to add a managed bean:

1. On the ManagedBean tab, click **Session** and then click **Add** to add a session scoped managed bean.
2. In the resulting window, click **Create a new Java class** for the managed bean.
3. A window opens and in that window, create a class named `UserManagedBean` in the `com.itso.example` package.
4. Go through the remaining windows in the managed bean wizard by clicking **Next** and then **Finish**.

Figure 3-10 shows the ManagedBean configuration after the wizard finishes.

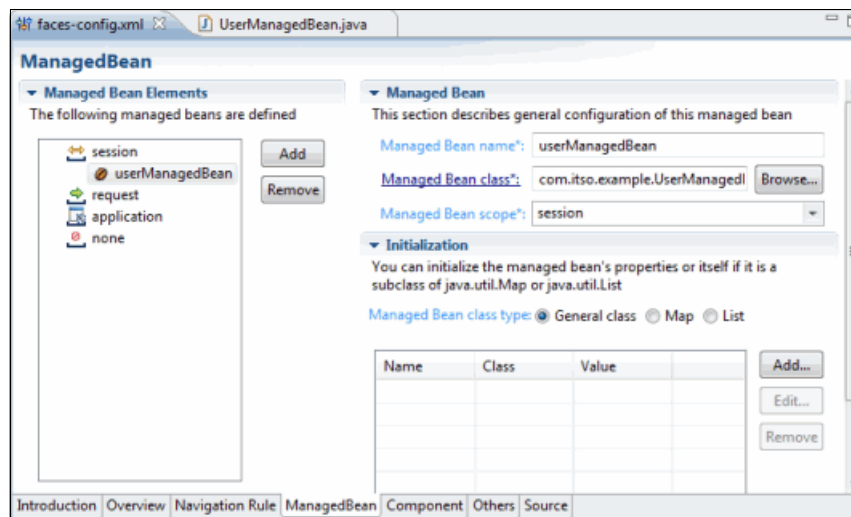


Figure 3-10 ManagedBean configuration for UserManagedBean

Now that the managed bean is added, complete the following steps to edit the `UserManagedBean` class:

1. Open the **UserManagedBean** class for editing.
2. Add String fields named `user` and `location`.
3. Add a getter and setter for both fields.
4. Add a method named, `clear` that sets both fields to null and returns the string "clear".
5. Add a method that is named `submit` that simply returns the string "submit".

Example code for the `UserManagedBean` class is shown in Figure 3-11 on page 68.

```
public class UserManagedBean {  
  
    private String name;  
    private String location;  
  
    public String getLocation() {  
        return location;  
    }  
    public void setLocation(String location) {  
        this.location = location;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String clear() {  
        setName(null);  
        setLocation(null);  
  
        return "clear";  
    }  
    public String submit() {  
        return "submit";  
    }  
}
```

Figure 3-11 Example `UserManagedBean` code

To further edit the properties of the managed bean, complete the following steps:

1. Go back to the Faces Config editor for the managed bean. You can set initial values for the managed bean's properties by clicking **Add** under the Initialization section.
2. From the property name drop-down menu, select the **Name property**. In the value field, enter Liberty User (Figure 3-12).

Property name*: name
Property class*: java.lang.String
Value type: value
Value: Liberty User

Figure 3-12 Adding a Managed Property

3. Repeat the process for the location property. In the value field, enter Liberty User Location.

Adding JSF-enabled JSP files

Create three JSP files, `input.jsp`, `visit.jsp`, and `goodbye.jsp` using the Create JSP wizard. To create each file, in the wizard's template selection page, select **New JavaServer Faces (JSF) Page (html)**.

The `input.jsp` file asks for two input parameters, a name and a location. The values are taken from the managed bean `userManagedBean`. Figure 3-13 shows example code to accomplish this process.

```
<f:view>
  <h2>Welcome, Liberty Developer </h2>

  <h:form>
    <BR>Enter your name:
    <h:inputText id="name" value="#{userManagedBean.name}"/>
    <BR>Enter your location:
    <h:inputText id="location" value="#{userManagedBean.location}"/>
    <BR>
    <h:commandButton action="#{userManagedBean.submit }" value="submit"/>
  </h:form>
</f:view>
```

Figure 3-13 Example code for `input.jsp`

The `visit.jsp` file outputs the name and location value from the `userManagedBean`. It then displays a button to return to the previous page, and a button to quit the application. Figure 3-14 shows example code for `visit.jsp`.

```
<f:view>
  <h:form>
    <h2>Hello, #{userManagedBean.name } from #{userManagedBean.location}</h2>
    <br>
    <h:commandButton action="#{userManagedBean.clear }" value="Go Back" />
    <br>
    <h:commandButton action="goodbye" value="Leave"/>
  </h:form>
</f:view>
```

Figure 3-14 Example code for `visit.jsp`

The `goodbye.jsp` file outputs the name value and has a command button that allows you to start over. Example code is shown in Figure 3-15.

```
<h:form>
  <h2>Goodbye, #{userManagedBean.name }</h2>
  <BR>
  <h:commandButton action="#{userManagedBean.clear}" value="Start Over"/>
</h:form>
```

Figure 3-15 Example `goodbye.jsp` file

Adding navigation rules and setting outcome properties

To add navigation rules, complete the following steps:

1. In the Faces Config editor, click the tab that is labeled **Navigation Rule**. Drag each of the three JSP files you created in the editor.
2. Create links between the files. If the palette is not already displayed, select **Window** → **Show View** → **Palette**. On the palette, click **Link**, and then click the `input.jsp` file. End the link by clicking the `visit.jsp` file. You should now see an arrow pointing from `input.jsp` to `visit.jsp`.
3. On the palette, click **Select** and then click the new link and view the properties in the properties view. If the properties view is not available, select **Window** → **Show View** → **Properties**. In the outcome field of the properties, enter `submit`.
4. Follow the same procedure to create a link between the `visit.jsp` file and the `goodbye.jsp` file, and set the outcome field to `goodbye`.

5. Create a link between the goodbye.jsp file and the input.jsp file, and set the outcome property to clear.
6. Finally, create a link from the visit.jsp file back to the input.jsp file and set the outcome to clear.

Figure 3-16 shows the navigation rule window after all the rules are created.

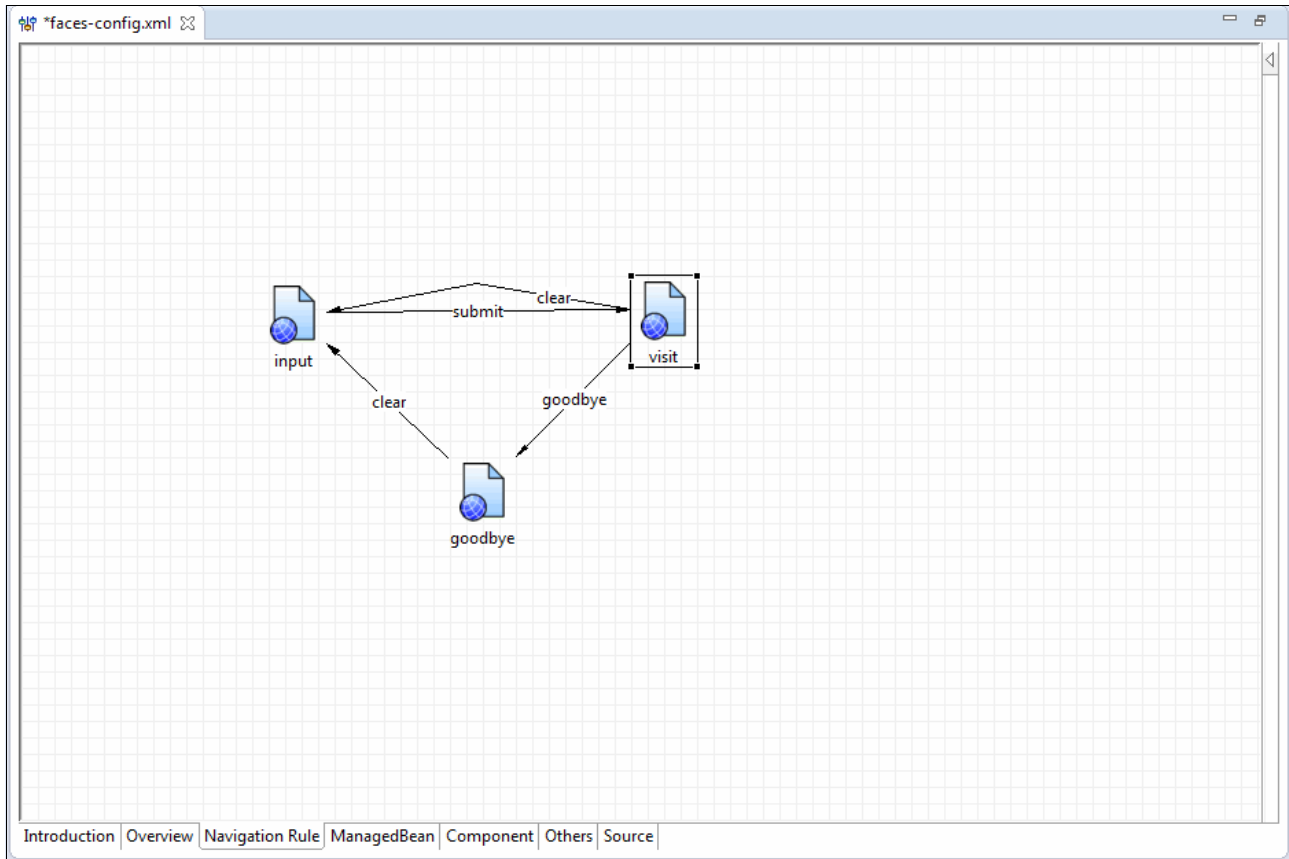


Figure 3-16 JSF navigation rules

Testing the application

Access the application by visiting the following website:

<http://localhost:9080/ITS0Web/faces/input.jsp>

Result: Notice how the managed bean fields were initialized without having to add code in the JSP. Also, notice that the application navigates to the correct pages based on the navigation rules without having to embed navigation logic in the JSP files.

3.1.4 Developing and deploying JAX-RS applications

The Java API for RESTful Web Services (JAX-RS) simplifies the development of applications that use Representational State Transfer (REST) principles.

More information about JAX-RS is available at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.mutiplatform.doc/ae/cwbs_jaxrs_overview.html

Building a simple JAX-RS application

To build and configure a JAX-RS application in the Liberty profile developer tools, complete the following steps:

1. To enable the JAX-RS facet on the web project, right-click the project and select **Properties**. Click **Project Facets** and enable JAX-RS by selecting the check box next to **JAX-RS (REST Web Services)**. And select 2.0 from the pull-down list next to **JAX-RS (REST Web Services)**, as shown in Figure 3-17.

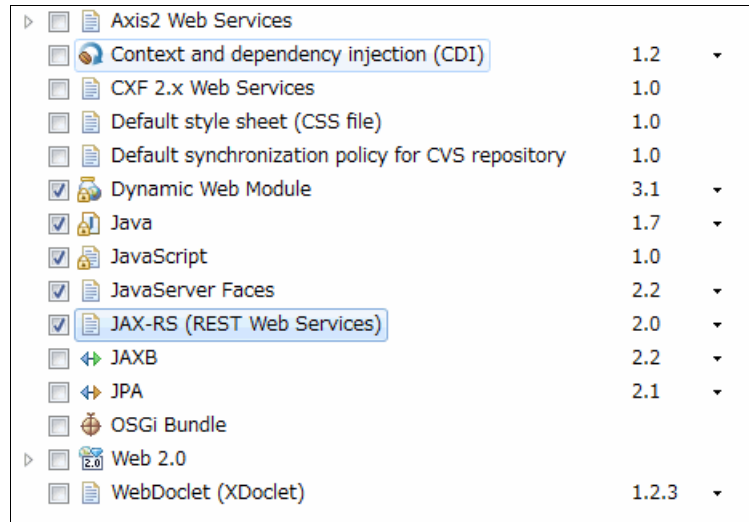


Figure 3-17 Web project facet list

2. After Enabling **JAX-RS (REST Web Services)**, you see a blue **i** mark below the window with a note that further configuration is available.
3. Clicking the note opens a window with various configuration options for JAX-RS. Enable the check box next to **Update Deployment Descriptor**. You can leave all of the values on this window as their defaults and click **OK**. If you open the web.xml file for the web project, you notice that a servlet and servlet mapping were added for JAX-RS.

Feature enablement

As with JSF, the Liberty profile server does not load the JAX-RS run time unless it is actively being used. Click **OK** on the facet window to enable the feature. If you load the server.xml file, in the source view, you notice that `<feature>jaxrs-2.0</feature>` now is added to the list of feature definitions.

Creating an application class

A JAX-RS application class is used to define the classes that are used in the application. Create one now by creating a new Java class that extends the `javax.ws.rs.core.Application`. The class should contain a `getClasses()` method that returns the classes that are involved in the application, as shown in Figure 3-18 on page 72.

```
public class ITSOJaxrsApplication extends Application {  
  
    @Override  
    public Set<Class<?>> getClasses() {  
        Set<Class<?>> classes = new HashSet<Class<?>>();  
        classes.add(ITSOJaxrsExample.class);  
        return classes;  
    }  
  
}
```

Figure 3-18 JAX-RS application class

Creating the ITSOJaxrsExample class

To create the class for your JAX-RS, complete the following steps:

1. Create a Java class named `ITSOJaxrsExample` or import the class that is included in the download material for this book. For more information, see Appendix A, “Additional material” on page 257.
2. The class must be annotated with `@Path(“/example”)` to indicate that the class is used with requests to the `ITSOWeb/jaxrs/example`.
3. Add a method that is called `getString` that returns a `javax.ws.rs.Response` object. The method must be annotated with `@GET`. Also, add a method that is called `postString` that takes a string as a parameter and returns a response object. The `postString` method should be annotated with `@POST`.

Figure 3-19 shows an example of the completed class.

```
@Path("/example")  
public class ITSOJaxrsExample {  
  
    @GET  
    @Produces(MediaType.TEXT_PLAIN)  
    public Response getString() {  
        return Response.ok("Hello ITSO from JAX-RS").build();  
    }  
  
    @POST  
    @Consumes(MediaType.TEXT_PLAIN)  
    @Produces(MediaType.TEXT_PLAIN)  
    public Response postString(String incomingMessage) {  
        return Response.ok("Hello, " + incomingMessage + " from JAX-RS").build();  
    }  
  
}
```

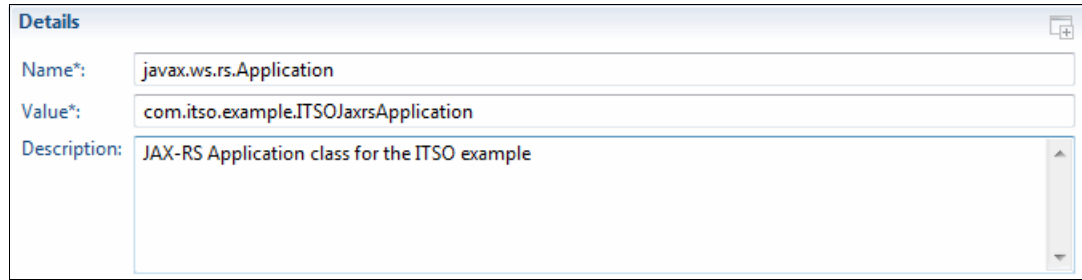
Figure 3-19 JAX-RS resource class

Adding a servlet initialization parameter

The servlet that was added when we updated the web project to enable the JAX-RS facet must know about your application class. This is done by creating an initialization parameter for the JAX-RS Servlet. To create an initialization parameter, complete the following steps:

1. Open the `web.xml` file in the design editor and right-click **JAX-RS Servlet**.
2. Select **Add** → **Initialization Parameter**.

3. On the resulting details window, enter `javax.ws.rs.Application` in the name field and the name of your Application class in the value field, as shown in Figure 3-20 on page 73.



The screenshot shows a 'Details' window with three input fields. The 'Name*' field is filled with 'javax.ws.rs.Application'. The 'Value*' field is filled with 'com.itso.example.ITSOJaxrsApplication'. The 'Description' field is filled with 'JAX-RS Application class for the ITSO example'.

Figure 3-20 Creating an initialization parameter for the JAX-RS servlet

Testing your application

To test your application, open a web browser and visit the following website:

`http://localhost:9080/ITSOWeb/jaxrs/example`

You should see the output `Hello ITSO from JAX-RS`.

Note: When you access web applications with JAX-RS enabled, you might see the error (shown in Example 3-1) displayed in the console output. This message does not represent a functional error and can be safely ignored.

Example 3-1 Error message with JAX-RS enabled

```
[ERROR ] An exception occurred during processing of the
org.apache.wink.server.internal.providers.exception.EJBAccessExceptionMapper
class. This class is ignored.
```

Creating the `ITSOJaxrsExampleClient` class

The Liberty profile supports Java EE Client API for JAX-RS (JAX-RS client) and provides the JAX-RS client wizard to create the JAX-RS client class.

To create the class for your JAX-RS client, complete the following steps:

1. Begin by opening **ITSOWeb** → **Services** → **REST**.

- Right-click **/example : com.itso.example.ITSOJaxrsExample**, and select **Generate** → **JAX-RS Client**, as shown in Figure 3-21.

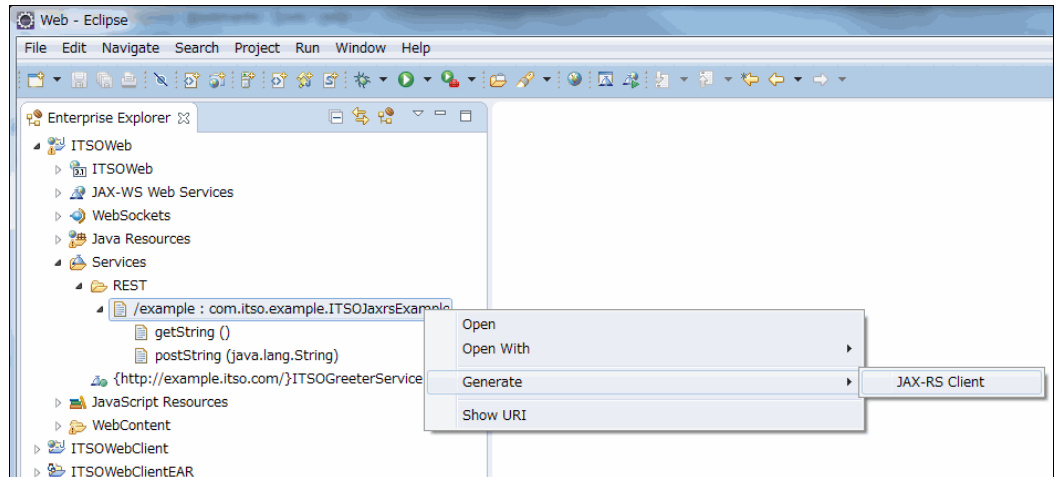


Figure 3-21 Open JAX-RS client wizard

- Use `ITSOJaxrsExampleClient` for the class name and `com.itso.example.client` for the package name, and enable **Generate JAX-RS Client Filter**. All other values can be left as default. Click **Finish**.

The window in Figure 3-22 shows the JAX-RS client wizard.

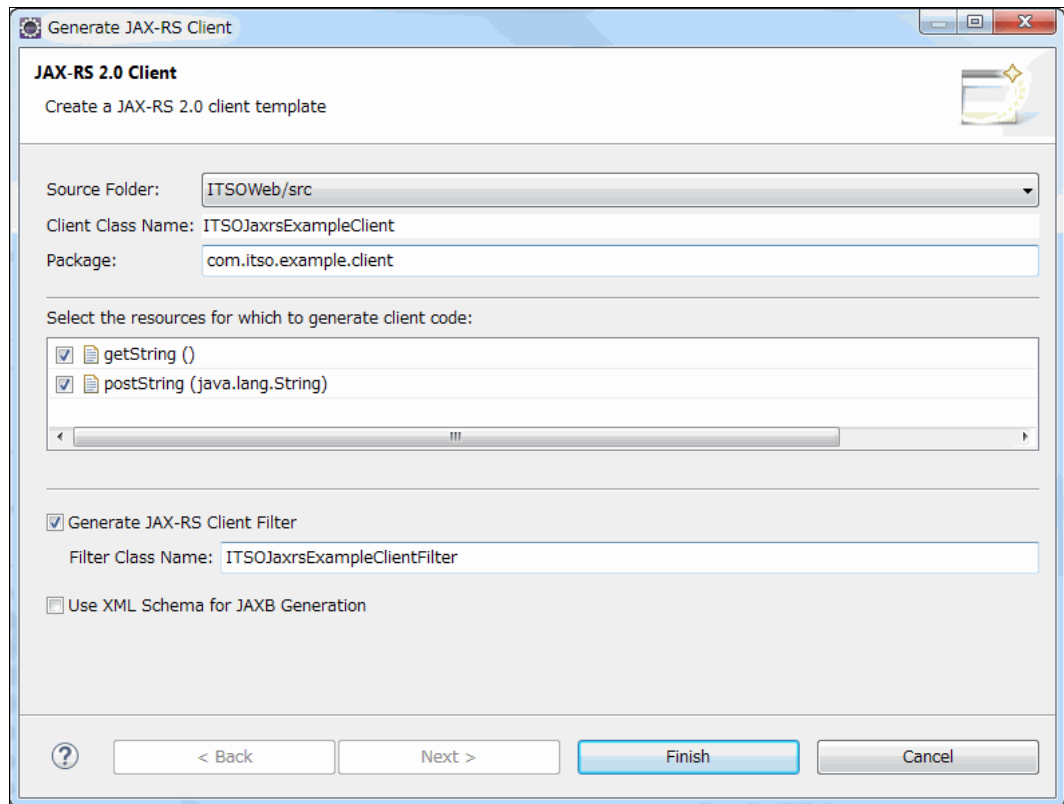


Figure 3-22 JAX-RS client wizard

- After finishing the wizard, you can verify that the `ITSOJaxrsExampleClient.java` file and the `ITSOJaxrsExampleClientFilter.java` file is included in the `ITSOWeb` project.

5. You do not have to enable the `jaxrsClient-2.0` feature because the `jaxrs-2.0` feature enables the `jaxrsClient-2.0` feature. However, if the Liberty profile uses only the JAX-RS client function, you have to enable the `jaxrsClient-2.0` feature.

Using JSON-P in the application

The Java API for JSON Processing (JSON-P) provides a standardized method for constructing and manipulating data to be rendered in JavaScript Object Notation (JSON). The Liberty profile supports JSON-P and provides the JSON-P feature.

You can use JSON-P in the RESTful services. In this part, we describe how you can use JSON-P in JAX-RS applications in the Liberty profile developer tools.

To use JSON-P in the `ITS0JaxrsExample` class, complete the following steps:

1. Open the `ITS0JaxrsExample.java` file, and find the `@GET` annotation.
2. The `@Produces` annotation under the `@GET` annotation is used to specify the MIME media types of representations. You should modify the `@Produces` annotation to remove `TEXT_PLAIN` and enter `APPLICATION_JSON`. Entering `APPLICATION_JSON` informs the client to accept the JSON data.
3. Next, you should modify the `getString()` method under the `@Produces` annotation to return JSON data to the client. You should create the JSON object using the `createObjectBuilder()` method, and add the values using the `add()` method, and return JSON object, as shown in Figure 3-23 as an example.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public JsonObject getString() {

    JsonObject jsonObj = Json.createObjectBuilder()
        .add("Name", "ITS0")
        .add("Location", "JSON-P")
        .add("Age", 30)
        .build();

    return jsonObj;
}
```

Figure 3-23 JSON-P example code

4. Enable the `jsonp-1.0` feature in server configuration.
5. To test the application, enter the link in your web browser:

`http://localhost:9080/ITS0Web/jaxrs/example`

You should see the JSON data was created by using JSON-P.

3.1.5 Using Context and Dependency Injection in web applications with the Liberty profile developer tools

By using the Context and Dependency Injection (CDI) feature of Java EE 6 and Java EE 7, you can inject resources into your applications. Previous versions of Java EE allowed you to inject certain resources, such as EJBs or a Persistence Context, but the CDI specification allows for more general-purpose resource injection.

CDI also helps manage the lifecycle of beans through annotations. Beans can be defined to live for the life of a request, session, application, or conversation.

For more information about CDI, refer to the following website:

http://www-01.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.zseries.doc/ae/cweb_cdi.html

Use the procedure that is outlined in the rest of this section to create a simple faces application that uses JavaBeans that are injected by CDI.

Enabling the CDI project facet

Enable the CDI project facet by completing the following steps:

1. Right-click the **ITSOWeb project** and select **Properties**. In the window that opens, click **Project Facets**.
2. Enable the CDI project facet by selecting the check box next to **Context and dependency injection (CDI)**, as shown in Figure 3-24 on page 76. This creates a named `beans.xml` in the web project. You can find it under **WebContent** → **WEB-INF**.

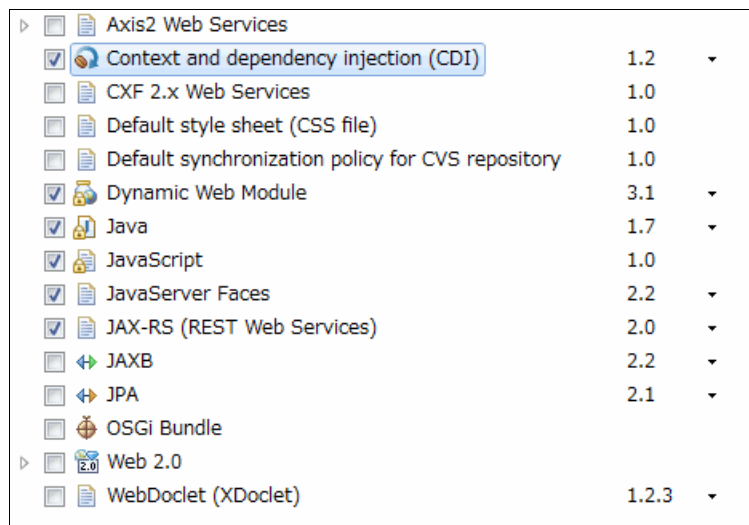


Figure 3-24 Enabling the CDI project facet

Creating a simple bean class

Create a Java class named `InjectedBean` or import the class that is included in the download material for this book. The class should be annotated with `@javax.enterprise.context.RequestScoped` to indicate that the bean's lifecycle is only as long as a single request.

The `InjectedBean` class should contain one method, `getHello()`, which simply returns the string "Hello from the injected bean". The completed class should look like the example in Figure 3-25.

```
@RequestScoped
public class InjectedBean {

    public String getHello() {
        return "Hello from the injected bean";
    }
}
```

Figure 3-25 The `InjectedBean` class

Creating another managed bean class

You can inject `InjectedBean` resources into another managed bean class by using CDI. Now, create a Java class named `ExampleBean` that uses the `InjectedBean` to return a value. Add the following actions to the class implementation:

- ▶ The `@javax.enterprise.context.RequestScoped` annotation should be added to the class to indicate the bean's lifecycle.
- ▶ The annotation `@javax.inject.Named("itsoBean")` should be added to the class to indicate that clients can access this managed bean by the name `itsoBean`.
- ▶ The class should have a field of type `InjectedBean`. The field should be annotated with `@javax.inject.Inject` to indicate that the value is injected by the container.
- ▶ The class should have a single method, `getMessage()` that simply returns the value of `injectedBean.getHello()`.

The completed class should look like the example in Figure 3-26 on page 77.

```
@RequestScoped
@Named("itsoBean")
public class ExampleBean {

    private @Inject
    InjectedBean injectedBean;

    public String getMessage() {
        if ( injectedBean != null ) {
            return injectedBean.getHello();
        }

        return "Something went wrong! The bean was not injected.";
    }
}
```

Figure 3-26 The `ExampleBean` class

Creating a JSP file to use the managed beans

Creating a JSP file to use the managed beans allows you to access the applications using CDI with web browser.

To create a JSP named `cdiExample.jsp`, use the Create JSP wizard. In the wizard's template selection window, click **New JavaServer Faces (JSF) Page (html)**.

In the body of the new JSP file, add an `outputText` element that has a value of `#{itsoBean.message}`, as shown in Example 3-2. Earlier, when "Creating another managed bean class", you annotated the `ExampleBean` class with `@javax.inject.Named("itsoBean")`, so the value for the `outputText` element is populated with the results of calling `getMessage()` on an `ExampleBean` instance.

Example 3-2 Outputting the bean value in `cdiExample.jsp`

```
<body>
<f:view>
    <h:outputText value="#{itsoBean.message}"/>
</f:view>
</body>
```

Testing the application

Test the web application by visiting the link:

<http://localhost:9080/ITSOWeb/faces/cdiExample.jsp>

Result: You should see the output (Hello from the injected bean). Your web application did not have to directly instantiate either the ExampleBean or the InjectedBean class.

3.1.6 Developing JAX-WS web services applications with the Liberty profile developer tools

The Java API for XML Web Services (JAX-WS) allows you to create web services applications in a Java EE environment. The JAX-WS API eases the development of web services applications by allowing developers to expose Java classes as web services using annotations. Developers are freed from having to deal with the complexities of the message format. For more information about JAX-WS, refer to the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/cwbs_jaxws.html

JAX-WS support is provided only by WebSphere Liberty profile editions other than the core edition.

Complete the actions outline in the following sections to expose a Java class as a web service in WebSphere Liberty profile.

Creating the service implementation

This example simulates bottom-up web services development where an existing implementation is exposed as a service. To do this, create a simple Java class to act as the service implementation.

Create a Java class in the ITSOWeb project named ITSOGreeter. It should have one method, named `getValue`, that takes a `String` as an input. It should return the `String` `Hello, <name>` from the ITSOGreeter. The completed ITSOGreeter class is shown in Figure 3-27.

```
public class ITSOGreeter {  
    public String getValue(String name) {  
        return "Hello, " + name + " from the ITSOGreeter";  
    }  
}
```

Figure 3-27 The ITSOGreeter class

Creating a web service

To expose the ITSOGreeter class as a web service, complete the following steps:

1. Begin by right-clicking the ITSOWeb project and selecting **New** → **Other**. In the “Select a wizard” dialog box, expand **Web Services** and select **Web Service**, as shown in Figure 3-28.

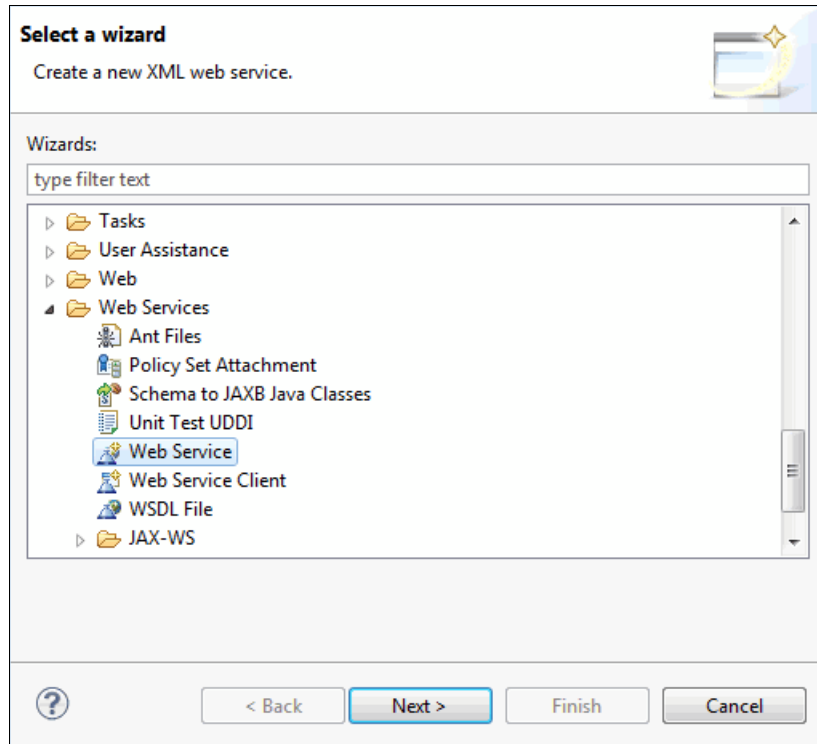


Figure 3-28 Web Service wizard selection

2. In the first window of the wizard, click **Browse** to select the ITSOGreeter class as the service implementation. Leave the web service type as the default, Bottom up Java bean Web Service.
3. On the left side of the window, there is a slider that lets you control how much of the web service is generated. For this example, leave the slider at the default value of **Start service**.
4. Under Configuration, the server run time should be WebSphere Application Server V8.5 Liberty Profile and the Web Service run time should be IBM WebSphere JAX-WS. These values are set by default, but if they are not, you can click the values to change them.

- By default, the wizard does not generate client code for the web service. You can change this by moving the lower slider up to **Start client**. As with the web service configuration, the server run time should be WebSphere Application Server V8.5 Liberty Profile and the Web Service run time should be IBM WebSphere JAX-WS. Figure 3-29 shows the completed wizard window.

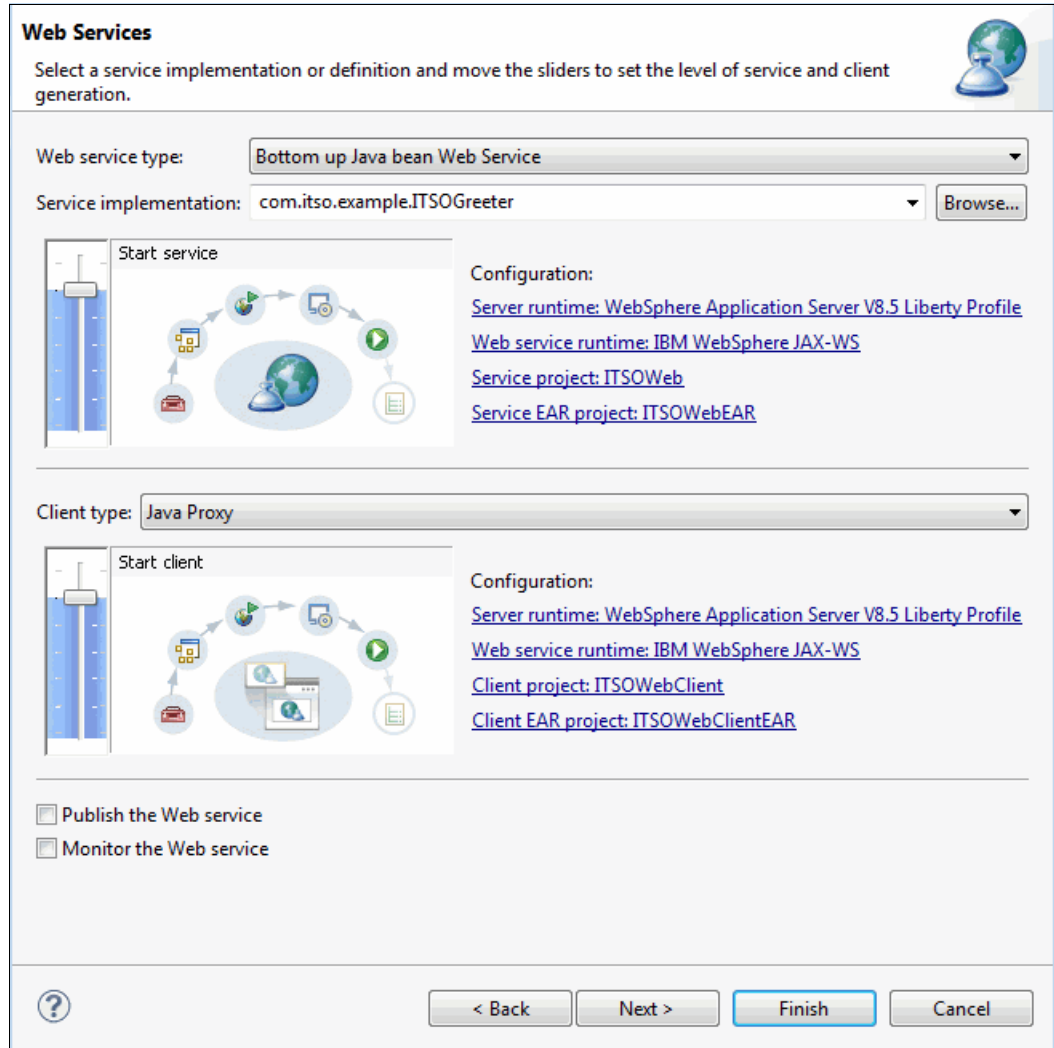


Figure 3-29 Web Services wizard

- Click **Next** to advance to the next window. The second window has many options for the generated web service. If you plan to use this example code as a starting point for the example in 6.6, “JAX-WS security” on page 200, select the check box next to **Generate WSDL file into the project**. This step is optional as the example in this chapter works with or without a generated WSDL file, and the file can be generated later if needed.
- Accept all of the defaults for other options and click **Next** to advance to the next window.
- Accept all of the defaults on the WSDL interface options window and click **Next**.
- If the Liberty profile server is not started, the next window requires you to start it before proceeding. After it is started, click **Next**.
- Accept all defaults on the web service client options window and click **Next**.

11. Click **Finish** to complete the wizard. Note the following changes that the wizard made:

- The wizard created a class named `ITSOGreeterDelegate` that provides a wrapper for the `ITSOGreeter` class. The `ITSOGreeterDelegate` class is annotated with `@WebService` to expose it as a JAX-WS web service.
- The wizard created a new EAR project named `ITSOWebEAR` and added the `ITSOWeb` dynamic web project to the new project's modules. It also added the new EAR to the server. You can verify this by loading the server configuration.
- A new web project named `ITSOWebClient` was created to contain the JAX-WS client code. In addition to the JAX-WS artifacts, the new project contains JAXB generated classes.
- A new EAR project named `ITSOWebClientEAR` was created to contain the `ITSOWebClient`. The wizard also deployed this EAR to the server.

Caution: The code in the client project is generated to use JAX-WS APIs that are not available in JavaSE 6. If you are using JRE Version 6, you might need to change the order of dependencies on the project's build path so that the Liberty profile runtime dependency precedes the JRE dependency.

Testing the web service

Test the web service by completing the following steps:

1. Make sure that the service side of the web service is working correctly by visiting the link:

`http://localhost:9080/ITSOWeb/ITSOGreeterService?wsdl`

This should return the WSDL document for the web service. Some web browsers hide XML documents by default. If you see an empty page when visiting the link, you might need to view the source to see the WSDL document.

2. Create a servlet in the `ITSOWebClient` project to drive the web service client. Using the Create Servlet wizard, create a servlet named `JAXWSGreeterClient`.

In the servlet's `doGet` method, create an instance of `ITSOGreeterService`. This is one of the client classes that was generated by the web services wizard. Call the `getITSOGreeterPort` method on the `ITSOGreeterService` instance to get an `ITSOGreeterDelegate` instance. Finally, print the value that is returned from calling `getValue` on the delegate instance.

Figure 3-30 on page 81 shows the completed servlet code.

```
@WebServlet("/JAXWSGreeterClient")
public class JAXWSGreeterClient extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PrintWriter writer = response.getWriter();
        ITSOGreeterService service = new ITSOGreeterService();
        String value = service.getITSOGreeterPort().getValue("JAX-WS Client");
        writer.println(value);
    }
}
```

Figure 3-30 `JAXWSGreeterClient` servlet code

3. Test the servlet by visiting the link:

`http://localhost:9080/ITSOWebClient/JAXWSGreeterClient`

Result: You should see the output Hello, JAX-WS Client from the ITSO Greeter.

3.1.7 Developing WebSocket applications with the Liberty profile developer tools

WebSocket is a standard protocol that enables a web browser or client application and a web server application to communicate by using one full duplex connection. You can develop the WebSocket 1.0 and WebSocket 1.1 applications that are supported by the Liberty profile V8.5.5.4 and later.

For more information about WebSocket, refer to the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.n.d.multiplatform.doc/ae/cwlp_websockets.html

Use the procedure that is outlined in the rest of this section to create a simple WebSocket application.

Creating a WebSocket server endpoint

Create a WebSocket server endpoint named `WebSocket` or import the class that is included in the download material for this book. In the case of creating, complete the following steps:

1. Begin with selecting the wizard by right-clicking the **ITSOWeb** project and selecting **New** → **Other**.
2. Select **WebSocket Endpoint**, and click **Next**. You can filter the entries that are shown in the dialog box by entering `web` in the search box, as shown in Figure 3-31 on page 82.

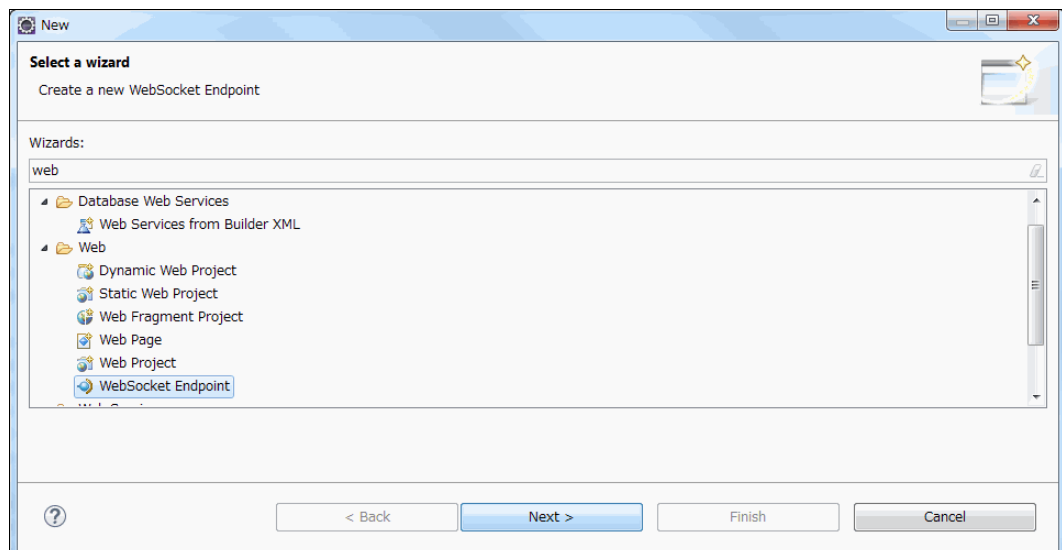


Figure 3-31 Selecting WebSocket Endpoint

3. Enter `com.itso.example` for the Java package and `WebSocket` for the Class name of the page, and click **Finish**. You can verify that `<feature>websocket-1.1</feature>` was added by opening the `server.xml` file.

- The `WebSocket.java` page automatically opens. The class should be annotated with `@OnOpen`, `@OnMessage`, `@OnClose`, and `@OnError`. The `onOpen()` method should contain the processing to store the `WebSocket` session for later use. And the `@OnMessage` method should contain the processing to close the `WebSocket` session and to send the messages. The completed class should look like the example in Figure 3-32 on page 83.

```
@ServerEndpoint("/WebSocket")
public class WebSocket {

    Session currentSession = null;

    @OnOpen
    public void onOpen(Session session, EndpointConfig ec) {
        currentSession = session;
    }

    @OnMessage
    public void receiveMessage(String name) {

        try {
            if (name.toLowerCase().equals("close")) {
                currentSession.getBasicRemote().sendText("Connection closed");
                currentSession.close();
            } else {
                currentSession.getBasicRemote().sendText("Hello, " + name + " from WebSocket");
            }
        } catch (IOException ex) {
        }
    }

    @OnClose
    public void onClose(Session session, CloseReason reason) {
    }

    @OnError
    public void onError(Throwable t) {
    }
}
```

Figure 3-32 `WebSocket` class

Creating `WebSocket` client endpoint

Create an html file as the `WebSocket` client endpoint named `WebSocketIndex` or import the html that is included in the download material for this book. In the case of creating, complete the following steps:

- Begin by creating the html file by right-clicking the **ITSOWeb** project and selecting **New** → **HTML File**.
- Enter `WebSocketIndex.html` for the File name of the page, and click **Next**.
- Select **New HTML File (5)** for the Templates, and click **Finish**.

- The `WebSocketIndex.html` automatically opens. Insert the JavaScript into the source editor between the body tags. Figure 3-33 on page 84 shows the html file after adding a form to submit a simple value to the WebSocket server endpoint and the processing for WebSocket. The html file should contain the processing for creating a WebSocket object instance and handling the lifecycle events of WebSocket.

```
<body>
<H2>Welcome to Liberty for Developers</H2>
  <div>
    <BR>Enter your name:
    <input id="name" type="text" />
  </div>
  <div>
    <input type="submit" value="Go" onclick="send()" />
  </div>
  <div id="messages"></div>
  <script type="text/javascript">
    var websocket = new WebSocket('ws://' + window.document.location.host + '/ITS0Web/WebSocket');

    websocket.onerror = function(event) {
      onError(event)
    };

    websocket.onopen = function(event) {
      onOpen(event)
    };

    websocket.onmessage = function(event) {
      onMessage(event)
    };

    function onMessage(event) {
      document.getElementById('messages').innerHTML += '<br />' + event.data;
    }

    function onOpen(event) {
      document.getElementById('messages').innerHTML = 'Connection established';
    }

    function onError(event) {
      alert(event.data);
    }

    function send() {
      var txt = document.getElementById('name').value;
      websocket.send(txt);
      return false;
    }
  </script>
</body>
```

Figure 3-33 `WebSocketIndex.html`

Testing the application

Test the web application by visiting the link:

`http://localhost:9080/ITS0Web/WebSocketIndex.html`

Result: You should be able to submit a value in the html file and see the output message from the WebSocket server endpoint that reflects the value that is given in the input field of the html file. And you should be able to repeat many times. If you submit “close” as the value, you should see the output (Connection closed).

3.1.8 Debugging applications with the Liberty profile developer tools

One of the most powerful features of developing in the Liberty profile developer tools environment is the ability to debug code running on the server directly from the development

environment. The following procedure is an example of how you debug a simple application in the Liberty profile developer tools environment.

To debug applications, complete the following steps:

1. The server must be run in debug mode. You can start (or restart) the server in debug mode by clicking the bug icon in the Servers view, as shown in Figure 3-34.

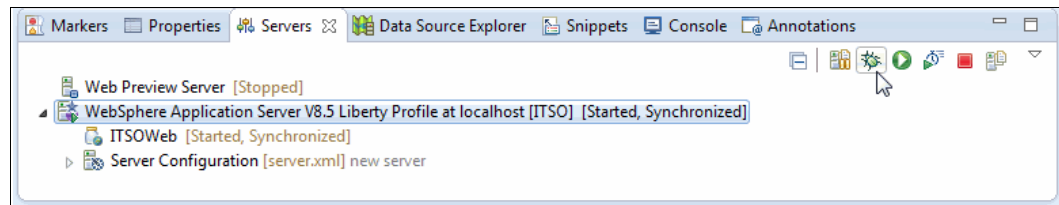


Figure 3-34 Restarting the server in debug mode

2. To debug the application, set a breakpoint in the **doGet** method of the `HelloITSO` servlet. On most platforms, you can do this by selecting a line in the method and pressing `Ctrl+Shift+b`. You can also right-click the vertical bar to the left of the line of code and select **Toggle Breakpoint**.
3. Open a web browser and enter the address for the `HelloITSO` example:
`http://localhost:9080/ITSOWeb/HelloITSO`
4. A dialog in the tools prompts you to switch to the debug perspective (unless you configured Eclipse to switch automatically). Click **YES**, and you see that the debugger hit the breakpoint in the **doGet** method. Make a change to the method and save the file. Advance past the breakpoint by clicking **Resume** on the menu bar (or by pressing `F8`.) The output in the browser changed based on the changes that you made.

For more information, refer to Chapter 7, “Serviceability and troubleshooting” on page 209.

3.2 Developing outside the Liberty profile developer tools

Developers who prefer working outside of the Liberty profile developer tools environment can also enjoy a simplified development experience by using the Liberty profile server. Liberty profile is designed to reduce development impact by providing a fast, lightweight environment for application development. The following list covers some of the key design features of the Liberty profile server that contribute to the new environment:

- ▶ Quick server start
- ▶ Hot deployment of applications
- ▶ Dynamic feature enablement
- ▶ Dynamic configuration
- ▶ Simple configuration that can be modified in a text editor

These characteristics of the Liberty profile ensure that developers can write, debug, and update applications without wasting time on server restarts or updating configuration through a complex administrative interface.

In this section, we cover several considerations to keep in mind when developing applications outside of the Liberty profile developer tools environment.

3.2.1 Feature enablement

The Liberty profile server only loads into memory the subset of the run time that is being used. You must specifically enable features in the configuration for them to be available to your applications. The following is a list of features that are used in the examples in this chapter:

servlet-3.1	Servlet 3.1
jsp-2.3	JavaServer Pages (JSP) 2.3
jpa-2.0	Java Persistence Architecture (JPA) 2.0
jaxrs-2.0	Java API for RESTful Web Services (JAX-RS) 2.0
jaxrsClient-2.0	Java EE Client API for JAX-RS
jsonp-1.0	Java API for JSON Processing
wasJmsClient-2.0	Java Messaging Service (JMS) Client 2.0
wasJmsServer-1.0	Java Messaging Service (JMS) Server 1.0
cdi-1.2	Context and Dependency Injection (CDI) 1.2
jaxws-2.2	Java XML Web Services (JAX-WS) 2.2
websocket-1.1	WebSocket

Some of those listed features also provide other features. For example, the `jsp-2.3` feature contains the `EL-3.0` feature. If your server configuration has `jsp-2.3` enabled, it does not need to have `EL-3.0` additionally specified to run servlets.

When you add or remove a feature, the run time is dynamically updated to enable or disable the function. You can test this by deploying a simple servlet and removing the `servlet-3.1` feature from the configuration. When you remove the feature, you see the text in Example 3-3 in the console output.

Example 3-3 Console output after removing the servlet feature

```
[AUDIT ] CWWKF00131: The server removed the following features: [servlet-3.1].
```

If you attempt to load a servlet, the server should fail to respond to the request. Now, enable the `servlet-3.1` feature again. You should see the output in Example 3-4 in the console.

Example 3-4 Console output after enabling the servlet feature

```
[AUDIT ] CWWKF00121: The server installed the following features: [servlet-3.1].
```

If you load the servlet now, the request succeeds.

3.2.2 Dynamic application update

By default, applications are automatically updated whenever a change to the application files is made. This allows you to save time that is not lost restarting applications manually.

This behavior can be seen by updating the `web.xml` file for a deployed web archive (WAR). Make a change, such as changing the value of a servlet mapping, and save the file. Then, load the servlet using the new mapping, and observe that the application is automatically updated.

3.2.3 Common development configuration

Liberty profile uses reasonable default values for configuration whenever possible. Some common cases where you might decide to update the configuration during development are covered next in this section.

Context root for web applications

Unless it is explicitly defined in the `server.xml` file or in the application, the context root for a web application is automatically determined from the file name of the WAR file. By default, the value of the context root is the WAR file name without the WAR extension. If you want to change the value of the context root, it can be specified in the configuration using the application definition that is shown in Example 3-5.

Example 3-5 An application definition from server.xml

```
<webApplication contextRoot="myWebAppRoot" location="ITS0Web.war" />
```

HTTP server ports

The default port for HTTP requests is 9080. You can also change this value in the configuration. To change the port to 9090, add the endpoint definition in Example 3-6 to the `server.xml` file. Setting the host value to `*` allows you to access this server from a different host machine. The default value for host is `localhost`, which limits access to browsers running on the local host machine.

Example 3-6 An endpoint definition from server.xml

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9090"/>
```

For more information, refer to 1.4.2, “Configuration-based application deployment” on page 28.

3.2.4 Dynamic configuration

Dynamic configuration updates in Liberty profile server mean that you do not have to waste time restarting the server when you make changes to the configuration. This section covers some scenarios that display the benefits of dynamic configuration updates.

Changing the HTTP server port

If port 9080 is in use on your machine or you are running multiple server instances, you might want to change the default HTTP port. With the server running, add the endpoint configuration, which is shown in Example 3-6, to your `server.xml` file. Save the file, and try to access a web application from the updated port. The server did not need to restart to make this change.

Adding a JNDI entry

Liberty profile server gives you the ability to bind JNDI entries in the configuration. These values are automatically updated when they are added, updated, or removed from the configuration.

Before using JNDI, you must add the JNDI-1.0 feature to your server’s configuration.

To test this, create a simple servlet and add code similar to that shown in Figure 3-35 on page 89.

```
try {
    Context ctx = new InitialContext();
    String value = (String) ctx.lookup("jndi/exampleValue");
    pw.println("<BR>Example value: " + value);
} catch (NamingException ex) {
    pw.println("Uh oh. I can't find the example value.");
}
```

Figure 3-35 Look up a value using JNDI

Run the servlet, and notice that the servlet was unable to find the value. Now, add the XML element in Example 3-7 to the `server.xml` file.

Example 3-7 XML element in `server.xml` file

```
<jndiEntry value="This is a sample value" jndiName="jndi/exampleValue"/>
```

Save the file and without restarting the server, run the servlet again. Notice that it now prints out the value that was set in the server configuration. You can also test updating the value or removing the `jndiEntry` element. In each case, the server is dynamically updated without a restart.

Caution: Only the server configuration is dynamically updated. Values are only read from `bootstrap.properties` when the server starts, so any value changes do not take place until you restart the server.

3.2.5 API JAR files

API JAR files for technologies that are included in the Liberty profile server are in the `dev` directory under the Liberty profile server home directory. The following list notes those directories:

- ▶ Java EE APIs are in `dev/api/spec`.
- ▶ IBM APIs are in `dev/api/ibm`.
- ▶ Third-party APIs are in `dev/api/third-party`.

The following list shows the API JAR files that are used for technologies that are covered in this chapter:

- ▶ Servlets
 `dev/api/spec/com.ibm.ws.javaee.servlet.3.1_1.0.9.jar`
- ▶ JSP
 `dev/api/spec/com.ibm.ws.javaee.jsp.2.3_1.0.9.jar`
- ▶ JSF
 `dev/api/spec/com.ibm.ws.javaee.jsf.2.2_1.0.9.jar`
 `dev/api/spec/com.ibm.ws.javaee.jsf.tld.2.2_1.0.9.jar`
 `dev/api/spec/com.ibm.ws.javaee.jstl.1.2_1.0.9.jar`
- ▶ JAX-RS
 `dev/api/spec/com.ibm.ws.javaee.jaxrs.2.0_1.0.9.jar`
 `dev/api/ibm/com.ibm.websphere.appserver.api.jaxrs20_1.0.9.jar`
 `dev/api/third-party/com.ibm.websphere.appserver.thirdparty.jaxrs_1.0.9.jar`

- ▶ CDI
dev/api/spec/com.ibm.ws.javaee.cdi.1.2_1.2.9.jar
- ▶ JMS
dev/api/spec/com.ibm.ws.javaee.jms.2.0_1.0.9.jar
- ▶ JAX-WS
dev/api/spec/com.ibm.ws.javaee.jaxws.2.2_1.0.9.jar

Caution: Only spec and ibm-api libraries are visible to applications by default. See 3.3, “Controlling class visibility in applications” on page 94 for more information about class visibility.

3.2.6 Debugging applications

You can use a Java debugger to debug your applications by starting the server in debug mode. To do this, start the server with the **server debug** command. The server waits for a Java debugger to attach on port 7777 before starting. Optionally, you can change the port that the server uses for debugging by setting the value of WLP_DEBUG_ADDRESS to the wanted port in your operating system’s environment.

3.2.7 Using Maven to automate tasks for the Liberty profile

Apache Maven is a build scripting tool that is designed to hide some of the complexities of build management from the user. Maven simplifies dependency management by maintaining a set of versioned artifacts in a repository.

Liberty profile enables building with Maven by providing a plug-in that can be used for common administrative tasks and a repository that contains certain API JAR files.

Plug-in repository

To use Maven for automating development tasks, first include the Liberty profile plug-in, in your project’s pom.xml file. The plug-in is in a public central repository. Example 3-8 shows a sample plug-in definition.

Example 3-8 Maven plug-in repository definition

```

<pluginRepository>
  <id>Liberty</id>
  <name>Liberty Repository</name>

  <url>http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/maven/re
pository/</url>
  <layout>default</layout>
  <snapshots>
    <enabled>>false</enabled>
  </snapshots>
  <releases>
    <enabled>>true</enabled>
  </releases>
</pluginRepository>

```

Specifying the plug-in and configuration

Example 3-9 shows how to load the Liberty Maven plug-in and specify a basic configuration. The configuration for `serverHome` and `serverName` is optional, but if it is not specified, it generally must be passed in as an argument when running from the command line.

Example 3-9 Loading and configuring the Liberty plug-in

```
<build>
  <plugins>
    <!-- Enable liberty-maven-plugin -->
    <plugin>
      <groupId>com.ibm.websphere.wlp.maven.plugins</groupId>
      <artifactId>liberty-maven-plugin</artifactId>
      <version>1.0</version>
      <!-- Specify configuration -->
    <configuration>
      <serverHome>/opt/IBM/wlp</serverHome>
      <serverName>testServer</serverName>
    </configuration>
  </plugin>
</plugins>
</build>
```

Examples: The following examples pass in the `serverHome` and `serverName` as arguments on the command line. These can be omitted if they are specified in the `pom.xml` file.

Creating a server

Use the following command to create a server named `SERVER_NAME` in the directory `SERVER_HOME`:

```
mvn liberty:create-server -DserverHome=SERVER_HOME -DserverName=SERVER_NAME
```

Deploying and undeploying applications

Use the following command to deploy an application that is in `APPLICATION_FILE` to the server named `SERVER_NAME`:

```
mvn liberty:deploy -DserverName=SERVER_NAME -DserverHome=SERVER_HOME
-DappArchive=APPLICATION_FILE
```

The server must be started to deploy the application.

Use the following command to undeploy the same application:

```
mvn liberty:undeploy -DserverName=SERVER_NAME -DserverHome=SERVER_HOME
-DappArchive=APPLICATION_NAME
```

The server must be started to remove the application. In this scenario, the `appArchive` parameter uses the application name rather than a file path.

Starting and stopping the server

Use the following command line to start a Liberty profile server instance:

```
mvn liberty:start-server -DserverHome=SERVER_HOME -DserverName=SERVER_NAME
```

Use the following command to stop a running server:

```
mvn liberty:stop-server -DserverHome=SERVER_HOME -DserverName=SERVER_NAME
```

Packaging and installing the server

Use the following command line to package an existing server named SERVER_NAME in a file named ARCHIVE_FILE:

```
mvn liberty:package-server -DserverName=SERVER_NAME -DserverHome=SERVER_HOME  
-DpackageFile=ARCHIVE_FILE
```

Use the following command to install a server from an archive named ARCHIVE_FILE:

```
mvn liberty:install-server -DassemblyArchive=ARCHIVE_FILE
```

Dependency repositories

The API JAR files that are in the dev/ibm-api directory underneath the Liberty Profile Server home directory are also available in a public Maven repository. To use the repository, add the repository definition in Example 3-10 to your pom.xml file.

Example 3-10 Maven repository definition

```
<repository>  
  <id>wlp.central</id>  
  
  <url>http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/maven/re  
pository</url>  
  <releases>  
    <enabled>>true</enabled>  
  </releases>  
  <snapshots>  
    <enabled>>false</enabled>  
  </snapshots>  
</repository>
```

Example 3-11 provides an example of referencing a specific artifact dependency. In this scenario, the example is referencing the IBM JAX-RS extension APIs.

Example 3-11 IBM JAX-RS extension APIs

```
<dependency>  
  <groupId>com.ibm.websphere.appserver.api</groupId>  
  <artifactId>com.ibm.websphere.appserver.api.jaxrs</artifactId>  
  <version>1.0</version>  
  <scope>provided</scope>  
</dependency>
```

Java EE dependencies, such as the servlet API, are not available in the IBM repository. These dependencies can be accessed by using the repository definition in Example 3-12.

Example 3-12 Repository definition

```
<repository>  
  <id>java.net</id>  
  <url>http://download.java.net/maven/2</url>  
</repository>
```

The dependency definition in Example 3-13 shows how to reference the Java EE 7API.

Example 3-13 Java EE 7API reference

```
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
  <version>7.0</version>
  <scope>provided</scope>
</dependency>
```

3.2.8 Using Ant to automate tasks for the Liberty profile

Liberty profile provides an Ant plug-in so you can automate build processes that include management of the server and applications.

Ant plug-in

The Liberty profile Ant plug-in is in the `dev/tools/ant` directory under the server home directory. To use the plug-in, you must make it available on the class path. To make the plug-in available, copy the plug-in JAR file to the `lib` directory of the Ant installation and reference it using an `antlib` namespace, as shown in Example 3-14.

Example 3-14 Using antlib to include the Ant plug-in

```
<project .... xmlns:wlp="antlib:com.ibm.websphere.wlp.ant">
  ...
</project>
```

Deploying and undeploying applications

You can automate the addition and removal of applications using the `deploy` and `undeploy` tasks that are shown in Example 3-15.

Example 3-15 Example of deploy and undeploy tasks

```
<wlp:deploy file="{basedir}/resources/ITS0Example.ear" timeout="40000"/>

<wlp:undeploy file="ITS0Example.ear" timeout="60000" />
```

Server administration

The following tasks are available for automating server administration tasks:

► Create

```
<wlp:server installDir="{wlp_install_dir}" operation="create"/>
```

► Start

```
<wlp:server installDir="{wlp_install_dir}" operation="start"
  serverName="{serverName}" />
```

► Stop

```
<wlp:server installDir="{wlp_install_dir}" operation="stop"
  serverName="{serverName}" />
```

► Status

```
<wlp:server installDir=${wlp_install_dir}" operation="status"/>
```

► Package

```
<wlp:server installDir="${wlp_install_dir}" operation="package"  
archive="packagedServer.zip"/>
```

3.3 Controlling class visibility in applications

Class visibility in the Liberty profile server is controlled by configuring a classloader for an application in the server configuration. Defining the classloader allows you to share common libraries with other applications and control the types of APIs that are visible to the application. Defining the classloader also ensures that applications load classes from their own libraries before using classes from the Liberty profile server run time.

3.3.1 Using shared libraries in applications

Java EE application development often involves using common utility JAR files across several applications. You can avoid having to maintain several copies of the same utility JAR files by using shared libraries.

A library, in the Liberty profile server, is composed of a collection of file sets that reference the JAR files to be included in the library. The library can be used in an application by creating a classloader definition for the application in the server configuration. The application classloader can share the in-memory copy of the library classes with other applications, or it can have its own private copy of the classes that are loaded from the same location.

Shared libraries, which are defined in the configuration, can be used only by applications that are explicitly configured in the `server.xml` file. Applications without configuration, such as those placed in the `dropins` directory, can use a global shared library, as described in 3.3.5, “Global libraries” on page 96.

3.3.2 Creating a shared library in the Liberty profile developer tools

To create a shared library in the Liberty profile developer tools, complete the following steps:

1. Load the `server.xml` file in the design editor.
2. Right-click the application and click **Add** → **Classloader**.
3. In the Classloader properties, click **New** next to Shared library references.
4. On the Shared Library details window, click **New** next to the Fileset reference to add a file set.
5. Enter the properties for the file set details window and save the `server.xml` file.

Look at the source view for the `server.xml` file. The application now contains a classloader, which contains a library, which contains a file set. Example 3-16 on page 95 shows the resulting configuration from the `server.xml` file.

Example 3-16 Shared library configuration in server.xml

```
<classloader >
  <commonLibrary>
    <fileset includes="utility.jar"/>
  </commonLibrary>
</classloader>
```

The Liberty profile configuration is flexible enough to allow many configuration elements to be specified either as a child of a parent element, or as an attribute reference from that parent element. The shared library configuration in 3.3.3, “Creating a shared library outside of the tools” on page 95 displays how some of the same configuration can be specified using references.

Caution: The Liberty profile developer tools use slightly different terminology than the server configuration file. A library in the tools is equivalent to a `privateLibrary` in `server.xml`. A shared library in the tools is equivalent to a `commonLibrary` in `server.xml`.

3.3.3 Creating a shared library outside of the tools

To create a shared library outside of the Liberty profile developer tools, you must manually edit the `server.xml` file to add the following items:

- ▶ A file set that references the JAR files on disk.
- ▶ A library that either contains or references the file set.
- ▶ A classloader for each application that uses the shared library. The classloader should reference the library element. If you want the application to share the in-memory copy of the library, use the `commonLibraryRef` attribute to reference the library. If you want the application to have its own copy of the library, use the `privateLibraryRef` attribute.

Example 3-17 shows the definition of a shared library that is used by two different applications. Both applications share an in-memory copy of the classes.

Example 3-17 Definition of two applications using the same shared library

```
<library id="loggingLibrary">
  <fileset dir="${server.config.dir}/logger" includes="ITSOLogger.jar"/>
</library>

<application name="ITSOApp1" location="ITSO_1.ear">
  <classloader commonLibraryRef="loggingLibrary"/>
</application>

<application name="ITSOApp2" location="ITSO_2.ear">
  <classloader commonLibraryRef="loggingLibrary"/>
</application>
```

3.3.4 Using libraries to override Liberty profile server classes

The Liberty runtime hides its internal Java packages from application classloaders and only exposes the defined APIs. Also, you should configure only the features that you require into your server. For these reasons, it is unlikely that you will need to override Liberty profile server classes, but this section describes how you can do that if you find it necessary. You can accomplish this by creating a shared library that contains the classes you want to use. Use the following options:

- ▶ In the Liberty profile tools

On the classloader properties window, change the value in the drop-down menu for delegation to parentLast.

- ▶ Outside the Liberty profile tools

To change the classloader delegation outside the Liberty profile developer tools, add the `delegation="parentLast"` attribute on the `classloader` element in the `server.xml` file.

3.3.5 Global libraries

The Liberty profile server allows you to provide a library or set of libraries that can be accessed by all applications. To add a JAR file to this global shared library, simply copy it to one of the following two locations under `#{WLP_USER_DIR}`:

- ▶ `shared/config/lib/global`
- ▶ `servers/server_name/lib/global`

The libraries in these directories are used by all applications that do not specifically define a classloader. If you define a classloader for an application, the global library classes are not available unless you add the library `global` as a shared library. The sample configuration in Example 3-18 shows an application that is configured to use the global shared library.

Example 3-18 Application using a global shared library

```
<application name="ITS0App1" location="ITS0_1.ear">
  <classloader commonLibraryRef="global"/>
</application>
```

3.3.6 Using a classloader to control API visibility

By default, an application in Liberty profile can see supported specification APIs (such as the Java EE Servlet specification) and IBM provided APIs. For the application to load classes from third-party libraries, such as those in `dev/third-party`, you must configure the allowed API types for the application classloader. In the Liberty profile developer tools, you can configure this value in the Allowed API types field on the classloader service details window. If you are directly editing the `server.xml` file, add the `apiTypeVisibility` attribute to the `classloader` element. In both cases, the value is to be a comma-separated list of any combination of the following items:

spec	Supported Specification Libraries
<code>ibm-api</code>	IBM provided APIs
<code>ibm-spi</code>	IBM provided SPIS
<code>third-party</code>	Third-party APIs

For example, the following classloader element allows an application to see specification libraries, IBM provided APIs, and third-party APIs:

```
<classloader apiTypeVisibility="spec,ibm-api, third-party"/
```




Iterative development of OSGi applications

In addition to Java EE applications, WebSphere Application Server Liberty Profile allows you to develop Open Services Gateway initiative (OSGi) applications. OSGi provides a framework for developing flexible, modular applications. With OSGi applications, you can deploy and manage applications as a set of versioned OSGi bundles.

The chapter contains the following sections:

- ▶ Introduction to OSGi applications in Liberty profile
- ▶ Developing OSGi applications in the Liberty profile developer tools
- ▶ Developing OSGi applications outside Liberty profile developer tools

4.1 Introduction to OSGi applications in Liberty profile

The Liberty profile server and Liberty profile developer tools help you to develop and deploy modular OSGi applications quickly. The advantages of OSGi are included in the following list:

- ▶ Reduces complexity through modular design.
- ▶ Integrates with standard Java EE technologies such as servlets.
- ▶ Promotes service-oriented design.
- ▶ Composes isolated enterprise applications using multiple versioned bundles with dynamic lifecycles.
- ▶ Provides careful dependency and version management.
- ▶ Offers declarative assembly of components.
- ▶ Allows sharing modules through a bundle repository that can host common and versioned bundles.
- ▶ Allows access to external bundle repositories.
- ▶ Allows administrative updates to deployed applications at a bundle level.

For more information, see the OSGi home page at the following website:

<http://www.osgi.org>

The OSGi support in the WebSphere Application Server Liberty profile is based on the Apache Aries open community project.

4.2 Developing OSGi applications in the Liberty profile developer tools

The Liberty profile developer tools make developing OSGi applications easier by providing graphical editors for OSGi artifacts such as OSGi Blueprint and manifest files. The tool environment allows you to safely create dependencies on other bundles without having to worry about problems being introduced by mistakes in manual editing of manifest files. The Liberty profile developer tools can also help with converting existing Java EE applications into OSGi applications.

4.2.1 Using the tools to build an OSGi application

In this section, we describe how to build a simple hello world style OSGi application using different bundles for the client, the service, and the API. The application shows a limited subset of what is possible with OSGi applications, and shows how to get started building OSGi applications in the Liberty profile developer tools.

Creating OSGi bundle projects

In this application, we use three bundles: a client bundle, a service bundle, and an API bundle. Begin by creating a project for each bundle by using the following process:

1. Click **File** → **New** → **OSGi Bundle Project**.
2. Enter a project name. This example uses the project names `ITS0.OSGiExample.API`, `ITS0.OSGiExample.Client`, and `ITS0.OSGiExample.Service`.
3. For the Target Runtime, select **WebSphere Application Server V8.5 Liberty Profile**.
4. Clear the **Add bundle to application** check box. If you leave this check box selected, a new OSGi application project is created.
5. Leave all other values as the default settings and exit the wizard by clicking **Next** → **Next** → **Finish**.

Starting with version 8.5.5.5, you can use the developer tools to create OSGi bundles using the Web 3.1 facet configuration as shown in Figure 4-1 on page 102.

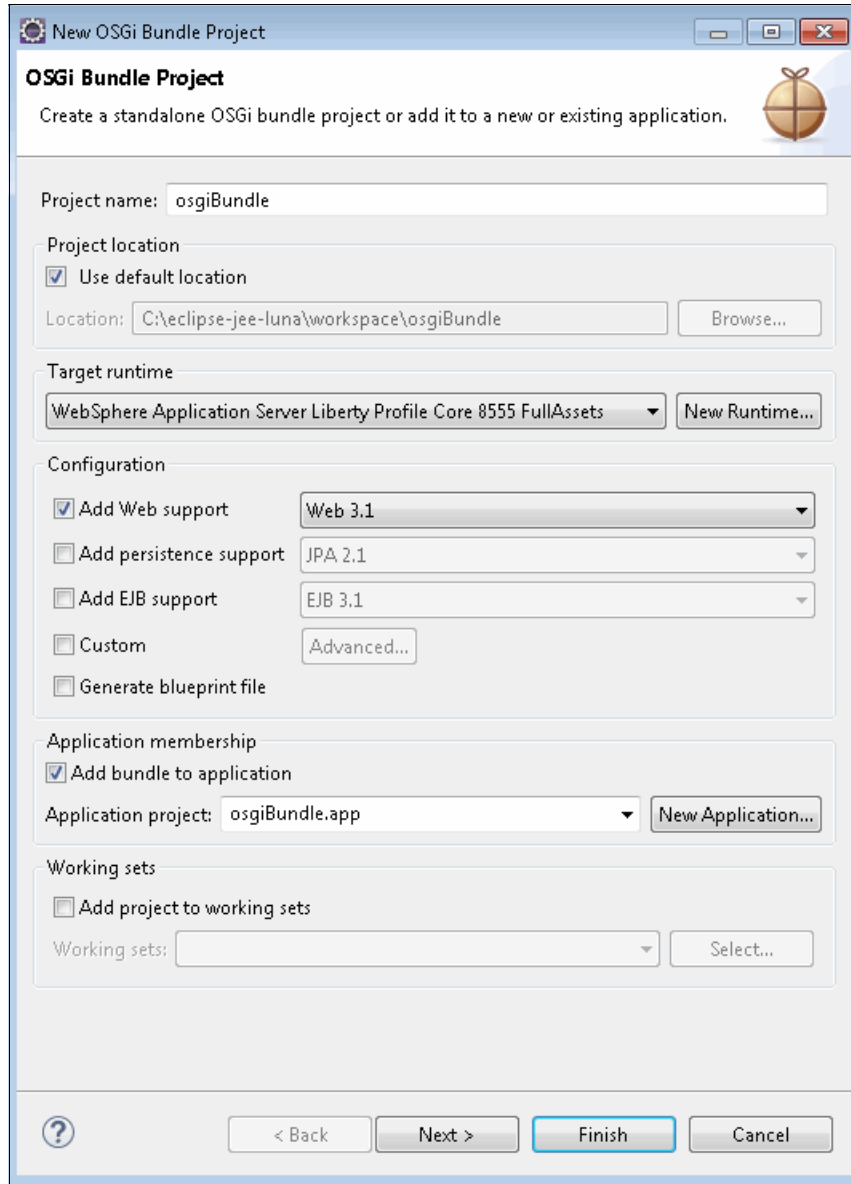


Figure 4-1 Creating an OSGi Bundle with Web 3.1

Creating the client API

In the `ITS0.OSGiExample.API` project, create an interface named `HelloAPI` in the `com.itso.example.osgi.api` package. The interface should contain a single public method with a void return type named `sayHello`.

Expose this interface class to other bundles by editing the manifest file for the project. You can open the manifest editor by double-clicking the Manifest element underneath the bundle project, as shown in Figure 4-2.

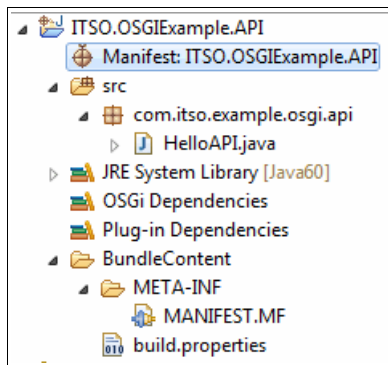


Figure 4-2 API bundle project

In the manifest editor, click the **Runtime** tab. In the exported packages pane, click **Add**. Select the `com.itso.example.osgi.api` package from the list and then click **OK**.

Creating the service bundle

The OSGi service bundle provides an implementation of the API. For the API interface to be visible in the service project, import the API package in the service bundle. To import the API package, complete the following steps:

1. Open the manifest editor for the `ITSO.OSGiExample.Service` project. In the Imported Packages pane of the Dependencies tab, click **Add**. In the package selection dialog, select `com.itso.example.osgi.api`. Save and close the manifest.
2. Create an implementation for the API by creating a new Java class in the `ITSO.OSGiExample.Service` project named `HelloImpl` in the `com.itso.example.osgi.service` package. The class should implement the `HelloAPI` interface. In the implementation of the `sayHello()` method, enter an output statement such as `System.out.println("Hello, Liberty developers from OSGi")`. Save and close the file.

Warning: When selecting an interface, you might need to start typing before the list of available interfaces is shown.

3. Export the service implementation using an OSGi Blueprint file. Begin by creating the file. Right-click the `ITSO.OSGiExample.Service` project and select **New** → **Blueprint File**. Leave all values as the defaults, and click **Finish**.

4. In the Overview pane of the Blueprint editor's design tab, click **Add**. Click **Bean** and click **OK**. Click **Browse** and select HelloImpl (you must start typing in the "choose type name" field to display the list of options). Leave the Bean ID field as the default value, HelloImplBean.
5. Click **Blueprint** in the Overview pane and click **Add**.
 - a. Click **Service**, and click **OK**.
 - b. Click **Browse** next to the Service Interface field and select **HelloAPI**.
 - c. Click **Browse** next to the Bean Reference field and select **HelloImplBean**.
 - d. Click **OK** and save the file.
 - e. Examine the source for the Blueprint file and verify that it resembles the XML that is shown in Figure 4-3.

```

<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <service id="HelloImplBeanService" ref="HelloImplBean"
    interface="com.itso.example.osgi.api>HelloAPI" />
  <bean id="HelloImplBean"
    class="com.itso.example.osgi.server>HelloImpl" />
</blueprint>

```

Figure 4-3 Service Blueprint file

Creating an OSGi client bundle

The OSGi client bundle calls the sayHello method on the service implementation. Similar to the service bundle, the client bundle must have access to the API by importing the API package in its manifest. Use the procedure from step 1 on page 103 to edit the manifest and import the API.

Create a HelloITS0Client class in the ITS0.OSGiExample.Client project in the com.itso.example.osgi.client package. The class should contain the following pieces:

- ▶ A private field of type HelloAPI named helloService.
- ▶ Get and Set methods for the helloService field.
- ▶ A method named **init** that is called when the client is initialized. This method should print a message and then run **helloService.sayHello()**.

The completed class should resemble the code that is shown in Figure 4-4.

```
package com.itso.example.osgi.client;

import com.itso.example.osgi.api.HelloAPI;

public class HelloITSOCient {

    private HelloAPI helloService = null;

    public void init() {
        System.out.println("The client is starting.");
        helloService.sayHello();
        System.out.println("The client is finished initializing.");
    }

    public HelloAPI getHelloService() {
        return helloService;
    }

    public void setHelloService(HelloAPI helloService) {
        this.helloService = helloService;
    }

}
```

Figure 4-4 Source code for HelloITSOCient

The client must be configured so that it can recognize the service implementation and have the service dependency that is injected into the helloService field. This is done by creating a Blueprint file for the client by using the following steps:

1. Create a Blueprint file by right-clicking the `ITSO.OSGiExample.Client` project and selecting **New** → **Blueprint File**. You can accept all default values for the file and click **Finish**.
2. Add a reference element to the Blueprint:
 - a. Click **Add** in the Overview pane in the Blueprint editor.
 - b. Click **Reference** and click **OK**.
 - c. Click **Browse** beside the Reference Interface field, and select **HelloAPI** from the list of classes.
 - d. In the Reference ID field, enter `helloRef`.
 - e. Click **OK** to finish adding the reference element.
3. Add a bean element to the Blueprint file:
 - a. Click **Add** in the Overview pane.
 - b. Select **Bean** and click **OK**.
 - c. Click **Browse**, select **HelloITSOCient**, and click **OK**.
 - d. Click **OK** again to finish adding the bean element.
 - e. In the bean properties window on the right side of the editor, locate the Initialization Method field and click **Browse**.
 - f. Select the `init()` method and click **OK**. This alerts the Blueprint container to run the `init` method when the `HelloITSOCient` class is instantiated.
4. Add a property to the `HelloITSOCientBean` by selecting the bean in the Overview pane and clicking **Add**. Select **Property** and click **OK**. In the Details window, enter `helloService` in the Name field. Then, click **Browse** beside the Reference field and select the `helloRef` reference. Click **OK** to finish adding the property.
5. Save the Blueprint file and switch to the Source tab.

The completed file should resemble Figure 4-5.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <reference id="helloRef"
    interface="com.itso.example.osgi.api.HelloAPI" />
  <bean id="HelloITSOClientBean"
    class="com.itso.example.osgi.client.HelloITSOClient"
    init-method="init">
    <property name="helloService" ref="helloRef"/>
  </bean>
</blueprint>
```

Figure 4-5 Blueprint file for `ITSO.OSGiExample.Client`

Creating an OSGi application

Create an OSGi application to contain the three OSGi bundles by completing the following steps:

1. Click **File** → **New** → **OSGi Application Project**.
2. Enter a project name, such as `ITSO.OSGiExample.App`. Leave the other values as the defaults and click **Next**.
3. In the contained bundles list, select all three OSGi bundle projects and click **Finish**.

4.2.2 Using the tools to deploy and test an OSGi application

To deploy the OSGi application to a WebSphere Liberty profile server instance, right-click the server in the Servers view and select **Add and Remove**. In the Add and Remove window, select the `ITSO.OSGiExample.App` application and click **Add**. Then, click **Finish** to complete adding the application to the server.

The application is now deployed and started. You should see output in the console similar to Figure 4-6.

```
[AUDIT ] CWWKG0016I: Starting server configuration update.
[AUDIT ] CWWKG0017I: The server configuration was successfully updated in 7.065 seconds.
[AUDIT ] CWWKF0012I: The server installed the following features: [localConnector-1.0, blueprint-1.0, servlet-3.1, wab-1.0].
[AUDIT ] CWWKF0013I: The server removed the following features: [servlet-3.0].
[AUDIT ] CWWKF0008I: Feature update completed in 6.963 seconds.
The client is starting.
Hello, Liberty developer.
The client is finished initializing.
[AUDIT ] CWWKT0016I: Web application available (default_host): http://localhost:9080/ITSO.OSGiExample.API/
[AUDIT ] CWWKT0016I: Web application available (default_host): http://localhost:9080/ITSO.OSGiExample.Service/
[AUDIT ] CWWKT0016I: Web application available (default_host): http://localhost:9080/ITSO.OSGiExample.Client/
[AUDIT ] CWWKZ0001I: Application ITSO.OSGiExample.App started in 4.610 seconds.
```

Figure 4-6 Output after adding the OSGi application to the server

The `localConnector-1.0`, `blueprint-1.0`, `servlet-3.1`, `wab-1.0` features are added automatically and the default `servlet-3.0` is removed automatically because it is superseded by the `servlet-3.1` feature. The `init` method of the client is called during the application start, so you see the messages that are printed in the `init` method in the console. Note also that the message from the service implementation is printed because the client started the `sayHello` method.

4.2.3 Adding an OSGi web application bundle

Now add an OSGi web application bundle to the OSGi application. The web application uses the existing service inside a servlet. Rather than using Blueprint to manage the service reference, use the service registry to locate the service. Complete the following steps:

1. Begin by creating an OSGi bundle project called `ITSO.OSGiExample.WebApp`. This process is similar to the process that is described in “Creating OSGi bundle projects” on page 101. The following list shows the differences:
 - a. On the first window of the create project wizard, select the **Add Web Support** check box.
 - b. Leave the check box checked for adding the bundle to the OSGi application so that it is added to the `ITSO.OSGiExample.App` application.
 - c. Optionally, on the web module window, enter a different value for the context root. In this example, we use a context root of `osgi`.
2. As was the case with the service and client bundles, you must edit the manifest for the web application bundle to import its dependencies. Open the manifest editor and navigate to the Dependencies tab. Notice that several servlet dependencies were automatically added to the manifest when the project was created. Add the `com.itso.example.osgi.api` package to the list of dependencies. Additionally, you must add `org.osgi.framework` version 1.5.0 to the list.
3. Create a servlet in the new bundle project by using the Create Servlet wizard. Name the servlet `HelloWABServlet`, use the `com.itso.example.osgi.wab` package name, and edit the servlet URL mapping value to be `/hello`. For more information about creating servlets in the Liberty profile developer tools, see 3.1.1, “Using the tools to create a simple servlet application” on page 60.
4. In the servlet, add a private field of type `HelloAPI` named `helloService`. In the `doGet` method, add a call to `helloService.sayHello()`.
5. Also, in the servlet, override the `init(ServletContext)` method. In this method, you use the `ServletContext` to get the `ServletConfig`. From the `ServletConfig`, get the `osgi-bundlecontext` attribute. This is the OSGi `BundleContext` object. You can use this `BundleContext` to look up the `HelloAPI` service in the service registry. The complete `init` method is shown in Figure 4-7.

```
@Override
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    ServletContext context = config.getServletContext();
    BundleContext ctx = (BundleContext) context.getAttribute("osgi-bundlecontext");
    ServiceReference ref = ctx.getServiceReference>HelloAPI.class.getName());
    helloService = (HelloAPI) ctx.getService(ref);
}
```

Figure 4-7 Example `init` method for the `HelloWABServlet`

6. Save the servlet.

You can test this by visiting a link such as the one below, providing your server is listening on localhost and port 9080 as defined in its `server.xml` configuration file:

`http://localhost:9080/osgi/hello`

You do not see any output in the browser (unless you added output in the servlet's `doGet` method) but you see the message `Hello, Liberty developer` displayed in the console output. This message shows that the servlet successfully retrieved the service from the service register in its `init` method and started the service's `sayHello` method in its `doGet` method.

A new feature *osgiAppIntegration-1.0* available with Liberty profile version 8.5.5.5, enables the integration of OSGi application services. This means that the OSGi applications running in the same Liberty profile server can share their services with each other. To accomplish this, you must do the following tasks:

1. Add the future *osgiAppIntegration-1.0* to the `server.xml` configuration file of your Liberty profile server.
2. Add the **Application-ExportService** header to the **META-INF/APPLICATION.MF** file of the application that needs to export the service as in Example 4-1.

Example 4-1 Application-ExportService header in *META-INF/APPLICATION.MF*

```
Application-ExportService: com.itso.example.osgi.service;binding:=local
```

3. Add the **Application-ImportService** header to the **META-INF/APPLICATION.MF** file of the application that needs to import the service as in Example 4-2.

Example 4-2 Application-ImportService header in *META-INF/APPLICATION.MF*

```
Application-ImportService: com.itso.example.osgi.service;binding:=local
```

An OSGi application can both export and import services from other OSGi applications from the same Liberty profile server by using both headers. The `binding:=local` directive has to be added at the end of the import and export headers so that the OSGi applications within the same Liberty profile server can communicate with each other.

4.2.4 Deploying an OSGi application to Liberty in the cloud

You can deploy an OSGi application in IBM Bluemix™ in order to share it and use the benefits of the cloud environment.

You can use IBM WebSphere Application Server Developer Tools to package a Liberty profile server to IBM Bluemix. The IBM Eclipse Tools for Bluemix needs to be installed in the Developer Tools so that you can define a Bluemix server that is required as a target to package your Liberty profile server. You also need a valid IBM Bluemix account.

To create an IBM Bluemix server profile by using your Developer Tools, select the **IBM Bluemix** option under the **IBM** group in the **New Server** window shown in Figure 4-8 on page 109.

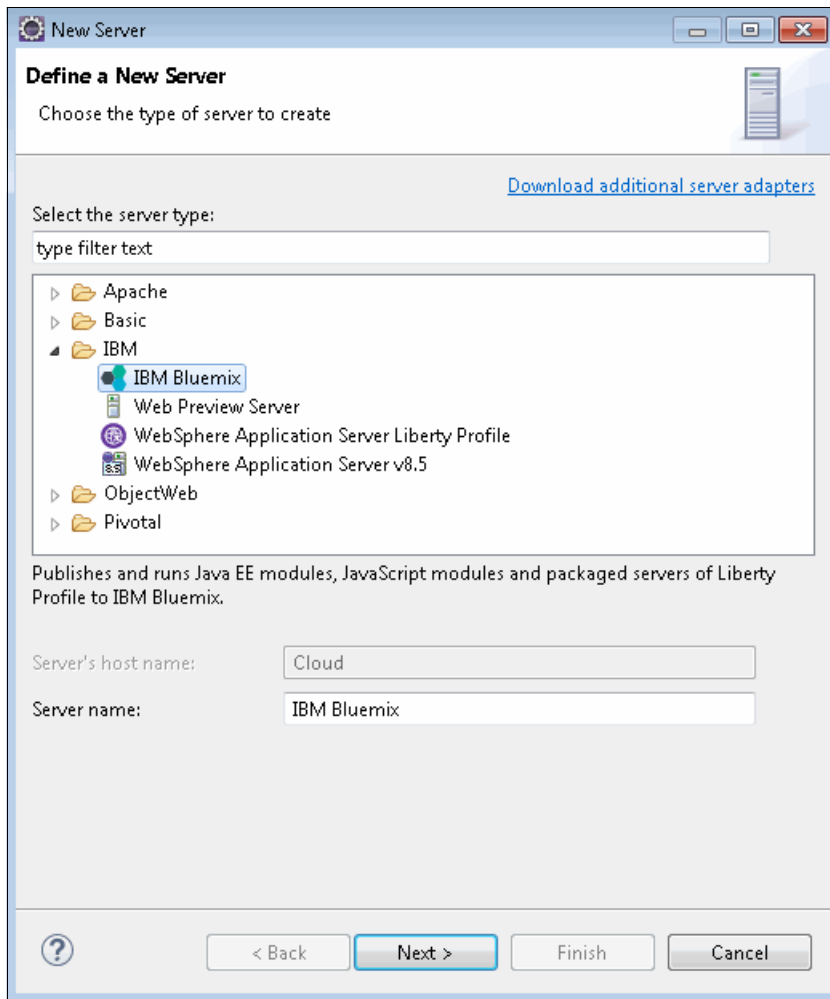


Figure 4-8 Defining a new IBM Bluemix server

To package a Liberty profile server that you already configured, right-click the server in the **Servers** view and select the **Package Server to IBM Bluemix** option that is located under **Utilities**, shown in Figure 4-9.

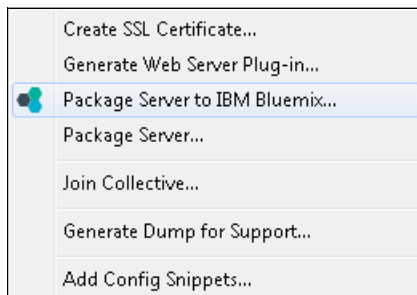


Figure 4-9 Package a Liberty profile server to IBM Bluemix

For more information about deploying applications with IBM Eclipse Tools for Bluemix, refer to the following website:

<https://www.ng.bluemix.net/docs/#manageapps/eclipsetools/eclipsetools.html#eclipse-tools>

4.3 Developing OSGi applications outside Liberty profile developer tools

Developers who prefer working outside of the Liberty profile developer tools environment can still develop OSGi applications quickly by using the Liberty profile server. In this section, we describe some considerations for developing in this environment and some tools to automate build and deployment processes.

4.3.1 Building and deploying OSGi applications outside of the tools

When building OSGi applications outside the Liberty profile developer tools, there are several API and SPI JAR files that are in the dev directory that contain important core classes, such as BundleContext and ServiceReference:

Note: For the following files, use the appropriate versions that are related to your Liberty profile environment.

- ▶ dev/api/spec/com.ibm.ws.org.osgi.cmpn.4.2.0_1.0.8.jar
- ▶ dev/api/spec/com.ibm.ws.org.osgi.core.4.2.0_1.0.8.jar
- ▶ dev/spi/third-party/com.ibm.wsspi.thirdparty.blueprint_1.1.8.jar
- ▶ dev/spi/spec/com.ibm.wsspi.org.osgi.cmpn.5.0.0_1.0.8.jar
- ▶ dev/spi/spec/com.ibm.wsspi.org.osgi.core.5.0.0_1.0.8.jar

Enterprise bundle archive (EBA) files can be placed in the dropins directory, such as EAR or WAR files. You can also define them explicitly using an application element in the server.xml file.

Important: To deploy OSGi applications to a Liberty profile server, you need to manually enable the features blueprint-1.0, osgi.jpas-1.0, and wab-1.0 in the server.xml configuration file.

OSGi console

You can use the OSGi console to debug your applications. The Liberty profile uses the Eclipse Equinox implementation of the OSGi core specification that provides this console. The OSGi console is not available by default; you have to enable it:

1. Add the osgiConsole-1.0 feature to your Liberty profile server.xml file.
2. Set the OSGi console port in the bootstrap.properties file of your Liberty profile server by adding the line `osgi.console=5471`.

The bootstrap.properties file has to be created manually in the same location as your server.xml file; it does not exist by default.

3. Restart your Liberty profile server.
4. Test the OSGi console by issuing a **telnet** command to the console port you have set.

OSGi bundle repositories

You can share common OSGi bundles both locally or remotely so that your OSGi applications can access them. This can be accomplished by editing the `server.xml` configuration file and adding a `bundleRepository` tag. Example 4-3 on page 111 shows a local repository configuration and Example 4-4 on page 111 shows a remote repository configuration type.

Example 4-3 Server.xml configuration for a local OSGi bundle repository

```
<bundleRepository>
  <fileset dir="directory_path" include="*.jar"/>
</bundleRepository>
```

The `directory_path` used to define a local OSGi bundle repository is the path to the directory containing the common OSGi bundles.

Example 4-4 Server.xml configuration for a remote OSGi bundle repository

```
<bundleRepository location="URL" />
```

The `URL` used to define a remote OSGi bundle repository defines the location of an OSGi Bundle Repository XML file and it supports HTTP, HTTPS, and file protocol types.

4.3.2 Using Maven to automate OSGi development tasks

You can automate the building and deployment of OSGi applications by using the Liberty Maven plug-in, with open source tools. An example of an open source tool is the EBA Maven plug-in that is developed by Apache Aries.

As described in 3.2.7, “Using Maven to automate tasks for the Liberty profile” on page 90, you can use the Liberty Maven plug-in to automate tasks. Some examples of these tasks are deploying applications and starting a Liberty profile server.

The EBA Maven plug-in allows you to build EBA archives from component bundles. It can automatically generate an Application Manifest file from information that is contained in the Maven pom file. For more information about the EBA Maven plug-in, refer to the following website:

<http://aries.apache.org/modules/ebamavenpluginproject.html>

4.3.3 Using Ant to automate OSGi development tasks

With Apache Ant and the Liberty Ant plug-in, you can automate the packaging and deployment of OSGi applications. Refer to 3.2.8, “Using Ant to automate tasks for the Liberty profile” on page 93 for more information about using the Liberty plug-in. This section gives further details about how to automate tasks, such as deploying and undeploying applications or starting and stopping servers.

You can use standard Ant tasks, such as `zip`, to package OSGi applications that can then be deployed using the Liberty plug-in. Example 4-5 shows sample syntax for creating an OSGi application named `ITS0ExampleApp` by packaging the `META-INF/APPLICATION.MF` file into the EBA file, and including all bundle files.

Example 4-5 Syntax for creating an OSGi application

```
<zip destfile="${output.dir}/ITS0ExampleApp.eba" basedir="${basedir}">
  <filename name="META-INF/APPLICATION.MF"/>
```

```
<fileset dir="${basedir}">
  <include name="*.jar"/>
</fileset>
</zip>
```

The Ant property `${output.dir}` is user-defined and specifies the output directory for your build. The Ant property `${basedir}` is predefined and specifies the directory that contains the Ant `build.xml` file.

Refer to the following website for more information about how to use Ant to automate tasks for the Liberty profile:

http://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/twlp_dev_ant.html?cp=SSEQTP_8.5.5%2F1-3-11-0-3-2-1-8&lang=en



Developing enterprise applications with Liberty profile

The Liberty profile server supports the full stack of Java EE technologies. In this chapter, you become familiar with advanced application features, such as accessing databases, using Enterprise JavaBeans, asynchronous programming with JMS, concurrent features, and more. By using these features, the Liberty profile developer tools provide code wizards and a server configuration editor to simplify development.

The chapter contains the following sections:

- ▶ Data access in the Liberty profile
- ▶ Developing Enterprise JavaBeans applications
- ▶ Developing Java Message Service applications
- ▶ Developing applications using asynchronous and concurrent features
- ▶ Developing applications using JavaMail

5.1 Data access in the Liberty profile

For data access, WebSphere Liberty profile supports the Java Persistence API (JPA) and JDBC. The Liberty profile developer tools simplify the development of data access applications for these technologies by assisting with the setup of required resources. In this section, we give an overview of data access in the Liberty profile and give examples of how to develop data access applications with the Liberty profile. We also briefly describe accessing noSQL databases from applications deployed in WebSphere Liberty.

5.1.1 Accessing data using a data source and JDBC

JDBC provides a database-independent API for querying and updating data. Liberty profile provides support for data access using 4.1 and earlier versions of the JDBC API.

Basic concepts for configuring data access in Liberty profile server

To use JDBC in an application, first configure access to the database. In Liberty profile, define a data source that has knowledge of the database. Several key configuration concepts important to this process are described in this section.

Files and file sets

A file set is simply a collection of files. They have several purposes in Liberty profile. In the context of data access, they are used to point to the JDBC driver library files provided by a database vendor.

Shared libraries

As described in 3.3, “Controlling class visibility in applications” on page 94, shared libraries provide a collection of classes that can be used by one or more applications or data sources. Libraries are composed of file set elements that define the location of the class files.

JDBC drivers

A JDBC driver definition references a shared library. You can optionally specify individual classes from the vendor-provided JDBC driver library to be used as the data source implementation. If you do not specify the implementation classes, the Liberty profile server infers the appropriate class name for most common databases.

Data sources

A data source is the key piece of configuration for data access in the Liberty profile server. In the simplest case, a data source is composed only of a JDBC driver and the JNDI name where the data source is made available. In more complex scenarios, you can specify other properties, such as the type of data source, transactional behavior, or vendor-specific database properties. For more information about the various configuration options that are available for data sources, see the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_dep_configuring_ds.html?cp=SSAW57_8.5.5%2F1-3-11-0-3-2-17-0-0

Adding a data source using Liberty profile developer tools

The Liberty profile developer tools facilitate the definition of adding a data source and related entities. The following scenario shows one way to define all of the artifacts that are necessary to add a data source to your application. Complete the following steps to add a data source:

1. Open the server.xml file for your WebSphere Liberty profile server. In the design view, select **Server Configuration** and click **Add**. Select **Data Source** and click **OK** (Figure 5-1).

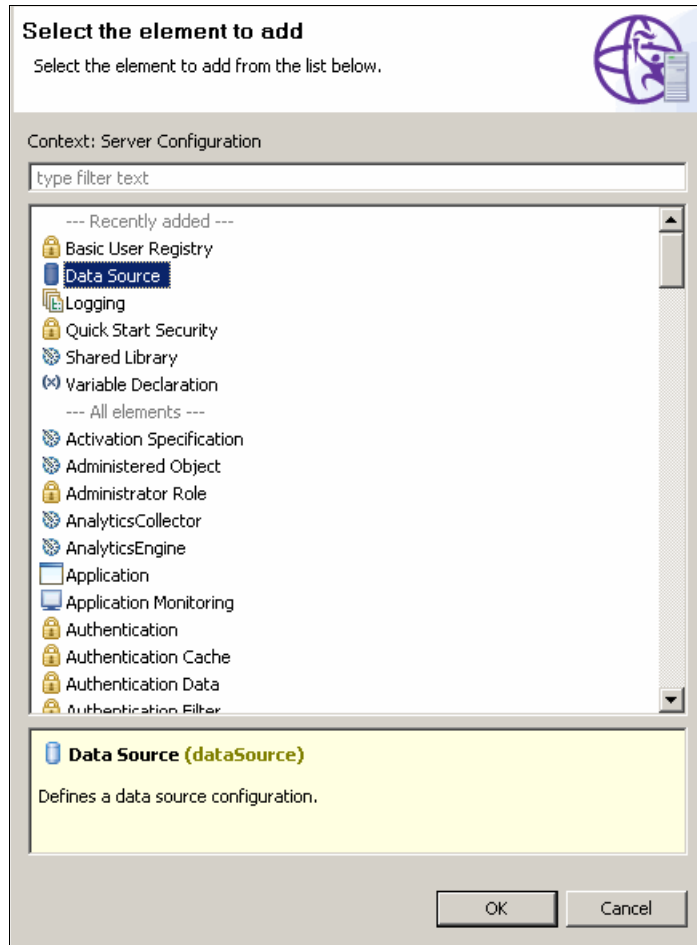


Figure 5-1 Adding a data source to an application

The **Enable Element** pop-up window is displayed and allows you to select the JDBC feature version. For Java EE 7 applications, select **jdbc-4.1**, as shown in Figure 5-2.

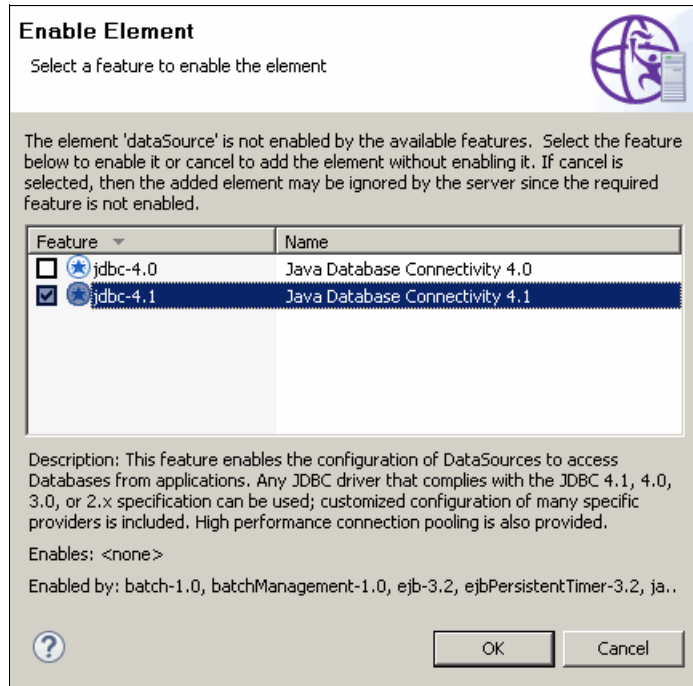


Figure 5-2 Selecting JDBC feature version

2. In the input field for JNDI Name, enter `jdbc/ITS0`.
3. The data source requires a JDBC Driver. On the Data Source Details window, beside the JDBC driver reference, click **Add**. You can add either *child* element or *global*. You should use global if you plan to use the same driver for many data sources. In this case, you will add child, which is a default, when you click **Add**. This opens the configuration window where you configure the new JDBC driver. See Figure 5-3.

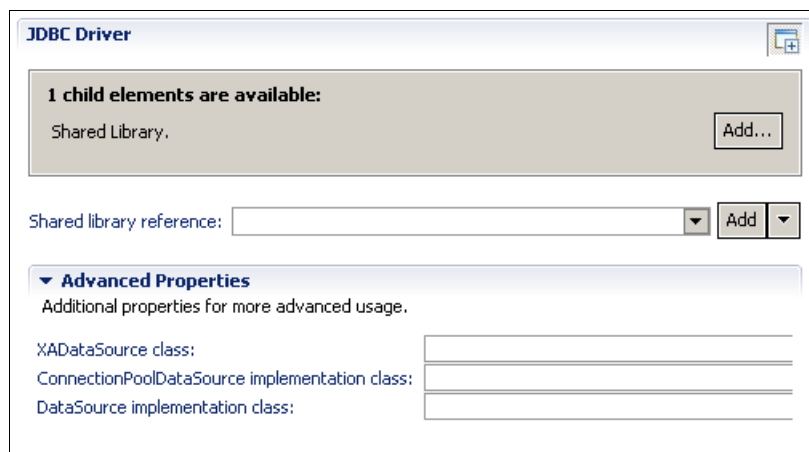


Figure 5-3 JDBC Driver details

- A JDBC driver requires driver implementation classes; this is specified using the shared library. Beside the Shared libraries reference field, click **Add** to create a shared library. This opens a window to configure the library. Enter `derbyLibrary` in the Name field, as shown in Figure 5-4.



Figure 5-4 Shared Library details

- Now, create a new file set for the shared library by clicking **Add** (located next to the input field for Fileset reference). This opens the Fileset Details window.

Finding the JAR files: The JAR files that are used for accessing your database are not provided as part of the Liberty run time. You need to select the Base directory in the Fileset details for the JAR files that are provided as part of your database run time. This example uses a Derby database, which can be downloaded from the following website:

<http://db.apache.org/derby>

Library placement: For easier Liberty server packaging, it is recommended to put driver JAR files in the Liberty shared resources folder (`WLP_HOME\usr\shared\resources`). This folder is referred to by default by the server variable `${shared.resource.dir}`. Variable usage is shown in the Base directory input field in Figure 5-5. We created the `derby` subdirectory in that folder and put the JAR file in that directory.

- Enter the directory where the JAR files for your JDBC driver are located (in the Base directory field for that JDBC driver). You can also click **Browse** to select a directory, as shown in Figure 5-5.

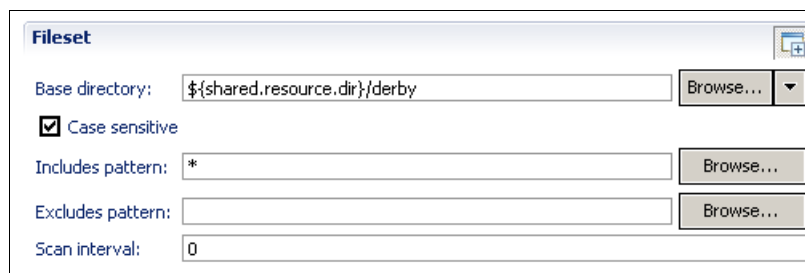


Figure 5-5 Fileset details

7. To finish the data source configuration, you need to provide database-related properties. Right-click the **Data Source** element under the Configuration Structure and select **Add** → **Derby Embedded Properties**. Then, use the following steps to finish the data source configuration:
 - a. In the Properties window, select **Create** in the Create database drop-down menu.
 - b. In the Database name field, enter the full path to database (for example, `c:\DerbyDatabases\ITS0`). If you provide just the database name, it will be created in the `WLP_HOME\usr\servers\serverName` directory.
8. Save the `server.xml` file. The data source is now available for applications.

Instead of using tools, configuring the data source can be achieved by editing the `server.xml` file directly. Example 5-1 shows part of the `server.xml` file that is related to the previous database configuration.

Example 5-1 Datasource definition in the server.xml

```
<dataSource jndiName="jdbc/ITS0">
  <jdbcDriver>
    <library name="derbyLibrary">
      <fileset dir="${shared.resource.dir}/derby"/>
    </library>
  </jdbcDriver>
  <properties.derby.embedded createDatabase="create" databaseName="ITS0"/>
</dataSource>
```

This scenario describes a simple data source configuration. More information about configuring database connectivity in the Liberty profile can be found at the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_dep_configuring_ds.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-17-0-0

Using the data source in an application

A data source that is defined in the server configuration can be accessed using JNDI. The `jndiName` property specifies the location of the data source in the namespace. To retrieve the `DataSource` object, simply look it up in JNDI. The code in Figure 5-6 displays how to accomplish this using the `javax.annotation.Resource` annotation on a field. You can also obtain the `DataSource` object by creating a new `InitialContext` object and starting the **lookup** method with the JNDI name of the data source as an argument.

```
@WebServlet("/DataSourceServlet")
public class DataSourceServlet extends BaseDataSourceServlet {
    private static final long serialVersionUID = 1L;

    @Resource(lookup = "jdbc/ITS0")
    private DataSource ds;

    protected DataSource getDataSource() {
        return ds;
    }
}
```

Figure 5-6 Retrieving a DataSource in a servlet

Defining a data source in an application

Applications can also define data sources through annotations or a deployment descriptor. To configure a data source this way, the application must have access to the JDBC driver classes. To do this, create a classloader for the application as described in 3.3, “Controlling class visibility in applications” on page 94. The classloader should reference a shared library that contains the JDBC driver classes.

Defining a data source using annotations

Example 5-2 defines a data source in a Java class of an application using annotations. The `DataSourceDefinition` annotation on the class defines various properties for the data source. Only the name and `className` are required properties.

Example 5-2 Annotations defining a data source

```
@DataSourceDefinition(name="java:comp/env/jdbc/ITS0DataSource",
    className = "org.apache.derby.jdbc.EmbeddedXADataSource40",
    databaseName = "ITS0",
    isolationLevel = Connection.TRANSACTION_READ_COMMITTED,
    loginTimeout = 88,
    maxPoolSize = 10,
    properties = {"connectionTimeout=0",
        "createDatabase=create",
        "queryTimeout=1m20s"})
```

JNDI Naming: The name for the embedded data source must be in one of the `java:global`, `java:app`, `java:module`, or `java:comp` name spaces.

Defining a data source using a deployment descriptor

You can define a data source in a WAR file deployment descriptor by adding a data source element to the `web.xml` file. You can simply edit the file, or use the deployment descriptor editor in the Liberty profile developer tools.

To create the data source in the tools, complete the following steps:

1. Begin by loading the `web.xml` file. In the design view, select the Web Application and click **Add**.
2. On the resulting window, select **Data Source**. Complete the properties for the data source. For example, enter `java:comp/env/jdbc/jdbc/ITS0DataSourceDD` for the name, `org.apache.derby.jdbc.EmbeddedDataSource` for the class name, and `ITS0DD` for the database name.
3. Now, click the new data source in the window on the left, and click **Add**. Select **Property** to add a property. In the property editor, enter `createDatabase` in the name field and `create` in the value field.
4. Save the deployment descriptor. If the server is running, the data source is now available to your applications.

Example 5-3 shows how the completed data source appears in the `web.xml` file.

Example 5-3 Data source definition in a deployment descriptor

```
<data-source>
  <name>java:comp/env/jdbc/ITS0DataSourceDD</name>
  <class-name>
    org.apache.derby.jdbc.EmbeddedDataSource
  </class-name>
```

```

<database-name>ITSODD</database-name>
<property>
  <name>createDatabase</name>
  <value>create</value>
</property>
</data-source>

```

The definitions here represent a small portion of the configuration options that are available for data sources. For more information about application-defined data sources, see the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_ds_appdefined.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-17-0-0-1

Using application-defined data sources

Obtaining an application-defined data source in an application uses the same procedure regardless of whether the data source was defined in a `DataSourceDefinition` annotation or in a deployment descriptor. In both cases, the data source is available in the namespace at the location you specified in the name property. Figure 5-7 shows sample code for retrieving a `DataSource` that was defined in the web application's deployment descriptor.

```

@WebServlet("/WebXmlDataSourceServlet")
public class WebXmlDataSourceServlet extends BaseDataSourceServlet {
    private static final long serialVersionUID = 1L;

    @Resource(lookup = "java:comp/env/jdbc/ITSODDataSourceDD")
    DataSource webXmlDataSource;

    @Override
    protected DataSource getDataSource() {
        return webXmlDataSource;
    }
}

```

Figure 5-7 Obtaining a `DataSource` defined in `web.xml`

Figure 5-8 shows sample code for accessing a `DataSource` that was defined in an annotation.

```

@WebServlet("/AnnotationDataSourceServlet")
@DataSourceDefinition(name = "java:comp/env/jdbc/ITSODDataSource",
    className = "org.apache.derby.jdbc.EmbeddedXADataSource40",
    databaseName = "ITSO", |
    properties = {"createDatabase=create"})
public class AnnotationDataSourceServlet extends BaseDataSourceServlet {
    private static final long serialVersionUID = 1L;

    @Resource(lookup = "java:comp/env/jdbc/ITSODDataSource")
    DataSource annotationDataSource;

    @Override
    protected DataSource getDataSource() {
        return annotationDataSource;
    }
}

```

Figure 5-8 Obtaining a `DataSource` defined in annotations

Testing the data sources

Appendix A, "Additional material" on page 257 contains an application and three prebuilt Derby databases that can be used to test each data source.

Caution: When importing entire projects from the additional materials, some of the examples for later sections might have compile errors. For example, the class `MongoServlet` has compile errors until you add the MongoDB JDBC driver to the project's class path, as described in "Configuring MongoDB in the server configuration" on page 134.

You can use one of the following database options during testing:

- ▶ To use the Derby databases, expand `ITS0_Derby.zip`, `ITS02_Derby.zip`, and `ITS0DD_Derby.zip` in the server's root directory. The data source definitions in the applications are designed to access these databases.
- ▶ To use another database, you must update the data sources to reference your database. Also, you must create a table named `USERTBL` in the database with the following configurations:
 - A column named `NAME` with a type that maps to `java.lang.String` (such as `VARCHAR`).
 - A column named `LOCATION` with a type that maps to `java.lang.String`.
 - A column named `ID` with a type that maps to `java.lang.Integer` (such as `INTEGER`). This column should be generated by the database.

The application contains three servlets: `DataSourceServlet`, `AnnotationDataSourceServlet`, and `WebXmlDataSourceServlet`. Each extends the abstract class `BaseDataSourceServlet`. Each servlet can perform four simple operations on the database table. The operations include selecting all rows, inserting a new row, dropping the table, or creating the table.

Note: Because of database differences, the create table statement might not work for all databases.

Each servlet can be tested using the JSP page `dataSource.jsp` that is included in the additional materials. To access the page, visit:

<http://localhost:9080/ITS0Web/dataSource.jsp>

Dynamic configuration updates for data sources

Changing the attributes of a data source element at run time results in dynamic updates to the server. Although the run time is updated immediately, some dynamic updates might affect a running server differently than others. For example, changing the `isolationLevel` attribute changes the isolation level for any future connection request, but current connections retain their existing isolation level. An update to the value of the `commitOrRollbackOnCleanup` attribute takes place immediately.

You can find a list of data source properties and their associated configuration update behaviors at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_ds_config_updates.html?cp=SSAW57_8.5.5%2F3-3-11-0-3-2-17-0-0-0

The rules that are noted apply only to data sources that are defined in the server configuration. When you update the properties of an application-defined data source, the entire data source is destroyed and the updated application uses a new data source. Effectively, from a developer perspective, all of the changes take place immediately.

5.1.2 Developing JPA applications

The Java Persistence API (JPA) provides a framework for working with relational databases in an object-oriented way. WebSphere Liberty profile server provides support for applications that use application-managed and container-managed JPA written to the JPA 2.1 or JPA 2.0 specification. In this chapter, we focus on the JPA 2.1 version. The support is built on top of EclipseLink with extensions to support the container-managed programming model.

Creating a JPA entity

The following procedure allows you to create a simple JPA application in the Liberty profile developer tools:

1. Before beginning, make sure that you defined a data source as described in 5.1, “Data access in the Liberty profile” on page 114.
2. Create a new Dynamic Web Project that uses the WebSphere Liberty runtime and the **3.1** web module version.
3. Begin by enabling the JPA facet on the created project; select the **2.1** version. This causes a `persistence.xml` file to be added to the project. If the project is deployed to the Liberty profile server, it also causes the tools to prompt you to add the JPA-2.1 feature to the server configuration.
4. Create a JPA entity class by right-clicking the project and selecting **New** → **JPA Entity**. Enter `com.ibm.itso.jpa21` in the Java package field and `Person` in the Class name field.

Figure 5-9 shows the first page of the JPA entity wizard.

The screenshot shows the 'Entity class' wizard dialog box. The title bar reads 'Entity class' and there is a small icon in the top right corner. Below the title bar, it says 'Create a new JPA entity. Only JPA enabled projects may be selected.' The dialog contains several input fields and buttons:

- Project:** A dropdown menu showing 'ITSOJPA21App'.
- Source folder:** A text field containing '{ITSOJPA21App}\src' and a 'Browse...' button.
- Java package:** A text field containing 'com.ibm.itso.jpa21' and a 'Browse...' button.
- Class name:** A text field containing 'Person'.
- Superclass:** An empty text field and a 'Browse...' button.
- Inheritance:** A section with two radio buttons: 'Entity' (selected) and 'Mapped superclass' (unselected). Below them is an 'Inheritance:' dropdown menu.
- XML entity mappings:** A section with a checkbox 'Add to entity mappings in XML' (unchecked) and a 'Mapping file:' text field with a 'Browse...' button.
- Navigation:** At the bottom, there are four buttons: a help icon (?), '< Back', 'Next >', 'Finish', and 'Cancel'.

Figure 5-9 New JPA Entity wizard

5. Click **Next** to advance to the next window. Under Table name, clear the **Use default** check box. Enter `PERSONTBL` in the Table name field.
6. Add an entity field by clicking **Add**. Click **Browse** next to the Type field to select `java.lang.Integer`, enter `id` in the Name field, and then click **OK**. **Select** the check box in

the Key column for the row containing the id field to indicate that the field is the primary key for the table.

7. Add an entity field by clicking **Add**. Click **Browse** next to the Type field to select `java.lang.String`, enter `name` in the name field, and then click **OK**.
8. Repeat step 7 to add a field named `location` and a `java.lang.String` type.

Figure 5-10 shows the completed second window of the JPA wizard. Click **Finish** to exit the wizard.

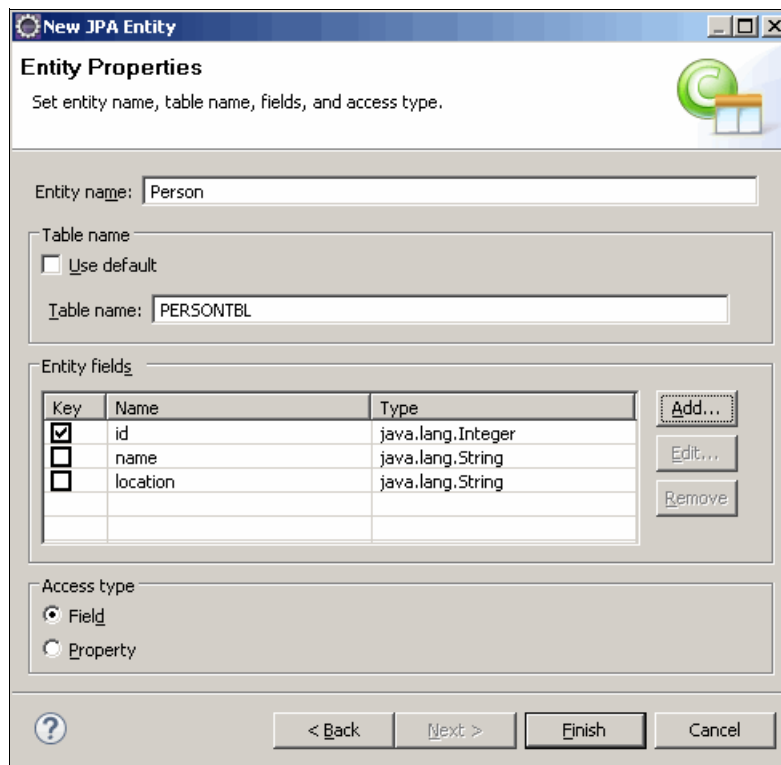


Figure 5-10 JPA Entity Properties window

9. Now, look at the source of the Person class. Notice that the `id` field is annotated with the `javax.persistence.Id` annotation to indicate that it is the primary key.
10. The Derby database that we use is automatically generating the `id` field. For JPA to handle the generated ID field correctly, it must be annotated with `javax.persistence.GeneratedValue`. The strategy attribute for the annotation should be set to `GeneratedValue.IDENTITY`. Figure 5-11 shows how the annotation should look.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer id;
```

Figure 5-11 The id field for the UserEntity class

Creating data access object

Once the JPA entities are created, they are managed using the `EntityManager`. It is a preferable practice to not embed data access code directly in your component (such as, servlet). Instead, create a data access object, which handles all the required tasks.

In this chapter, two possible solutions for data access are presented and noted in the following list:

- ▶ POJO-based data access object (DAO), which uses local transactions and CDI.
- ▶ EJB-based data access object, which uses JTA container-managed transactions.

Because Java EE 6 and later allows embedding EJBs in the web module, it is now easier to use EJBs for data access and to let the container manage transactions.

Creating a POJO-based DAO

The following process shows you how to create and configure POJO-based JPA entity management.

Create a class to handle the actual persistence by right-clicking the project and selecting **New** → **Class**. Enter `com.ibm.itso.jpa21` for the package name and `PersonPojoDao` for the class name. In the source editor, create two methods: `persistPerson` and `listPersons`.

The class is using `@ApplicationScoped` annotation to make the instance of the `PersonPojoDao` available in the application with injection and `@Named` annotation to make it accessible in JSP pages by using Expression Language (EL). `PersonPojoDao` is also using `@PersistenceUnit` to inject the `EntityManagerFactory` object. In the following simple example, there is no error handling logic needed.

The `listPersons` method runs a query that selects all `Person` objects and returns the results.

In the `persistPerson` method, you start the transaction and persist a new `Person` object. Finally, you have to commit the transaction or engage rollback in case of an error.

The complete class is presented in Example 5-4.

Example 5-4 PersonPojoDao class

```
@ApplicationScoped
@Named("personDao")
public class PersonPojoDao {
    @PersistenceUnit
    private EntityManagerFactory emf;

    public Integer persistPerson(Person person) {
        EntityManager em = emf.createEntityManager();

        try {
            em.getTransaction().begin();
            em.persist(person);
            em.getTransaction().commit();
            System.out.println("Created Person: id=" + person.getId() + " name=" +
person.getName());
            return person.getId();
        }
        finally {
            // clean up in case of exception
            if (em.getTransaction().isActive()){
                em.getTransaction().rollback();
            }
            em.close();
        }
    }
}
```

```

public List<Person> listPersons() {
    EntityManager em = emf.createEntityManager();

    try {
        em.getTransaction().begin();
        List<Person> resultList = em.createQuery("Select p from Person p",
Person.class).getResultList();
        em.getTransaction().commit();
        System.out.println("List returned: " + resultList.size());
        return resultList;
    }
    finally {
        // clean up in case of exception
        if (em.getTransaction().isActive()){
            em.getTransaction().rollback();
        }
        em.close();
    }
}
}
}

```

Configuring a persistent unit for local transactions

To allow the application to connect to a database, you need to configure a persistent unit by using the following steps:

1. Open the persistence.xml file in the design editor. Then, select the **Connection** tab.
2. Select **Resource Local** from the Transaction type combo and enter jdbc/ITSO in the Non-JTA data source field, as shown in Figure 5-12.

Figure 5-12 Persistence unit connection configuration

Creating an EJB-based DAO

The following steps show you how to create and configure EJB-based JPA entity management:

1. Create an EJB class to handle the actual persistence by right-clicking the project and selecting **New** → **Other**. In the **Select a wizard** pop-up window, start typing `Session` and select **Session Bean**, as shown in Figure 5-13.

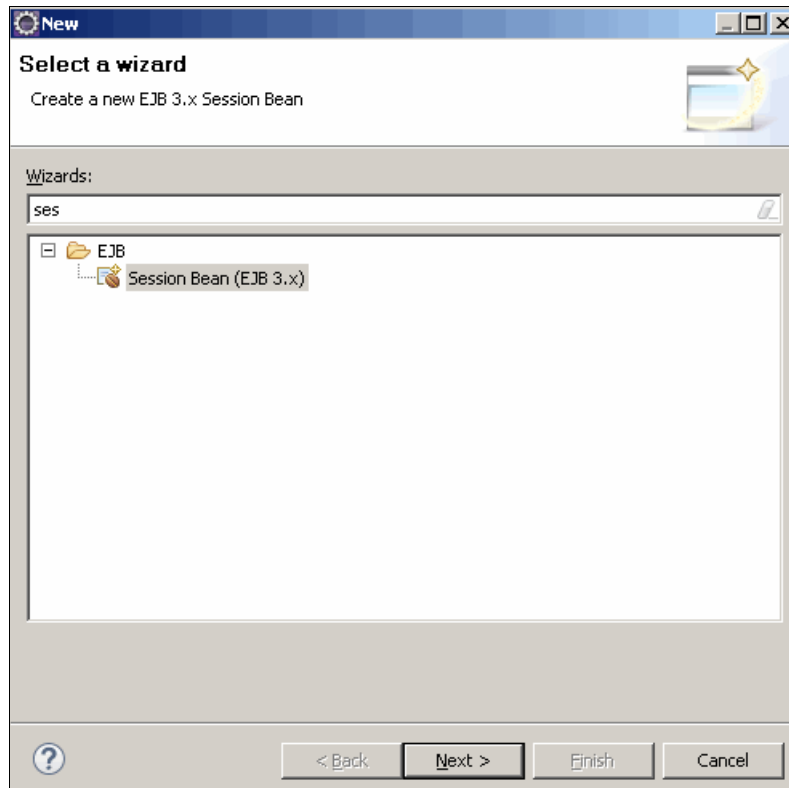


Figure 5-13 Selecting Session Bean in wizard

2. Enter `com.ibm.itso.jpa21` for the package name and `PersonEJBDao` for the class name. In the source editor, create two methods: `persistPerson` and `listPersons`.

The class is using `@Stateless` annotation to mark the class as EJB and `@Named` annotation to make it accessible in JSP pages by using EL. `PersonEJBDao` is using `@PersistenceContext` annotation to inject `EntityManager`. There is no error handling logic in the following simple example.

The `listPersons` method runs a query that selects all `Person` objects and returns the results. The `persistPerson` persists a new `Person` object. The complete class is presented in Example 5-5.

Example 5-5 PersonEJBDao class

```
@Stateless
@LocalBean
@Named("personEJBDao")
public class PersonEJBDao {
    @PersistenceContext
    private EntityManager em;

    public Integer persistPerson(Person person) {
```



```

        em.persist(person);
        em.flush();
        System.out.println("Created Person: id=" + person.getId()
            + " name=" + person.getName());
        return person.getId();
    }
    public List<Person> listPersons() {
        List<Person> resultList =
            em.createQuery("Select p from Person p",
                Person.class).getResultList();
        return resultList;
    }
}

```

Configuring persistent unit for JTA transactions

1. Open the persistence.xml file in the design editor. Then, select the **Connection** tab. Select **Default (JTA)** from the Transaction type combo and enter jdbc/ITS0 in the JTA data source field, as shown in Figure 5-14.

Figure 5-14 Configuring persistent unit connection details

Creating the front end for testing JPA

In this section, to test JPA-enabled classes, you create both a simple front-end application and a JSP page and a servlet. Use the following steps to complete the process:

1. Begin by creating a JSP file named `index.jsp`. The JSP contains a table with a list of persons, a form with two input fields, and a Submit button. The page is accessing a dao object using the name defined in `@Named` annotation. Example 5-6 shows the completed JSP file.

Example 5-6 Complete index.jsp file

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <!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>Hello Liberty JPA</title>
</head>
<body>
<H2>Hello, Liberty Developer from JPA.</H2>
These are our previous visitors:
<TABLE border=5>
<tr>
    <TH>ID</TH><TH>Name</TH><TH>Location</TH>
</tr>
<!-- change to items="${personDao.listPersons()} to use Pojo dao -->
<c:forEach var="person" items="${personEJBDao.listPersons()}">
    <tr>
        <td><c:out value="${person.id}"/></td>
        <td><c:out value="${person.name}"/></td>
        <td><c:out value="${person.location}"/></td>
    </tr>
</c:forEach>
</TABLE>
<form action="JPA21Servlet" method="POST">
Name:<input type="text" name="name"/><BR>
Location: <input type="text" name="location"/><BR>
<input type="submit" value="Submit New Visitor"/>
</form>
</body>
</html>
```

-
2. Create a servlet named `JPA21Servlet`. The servlet uses annotations to either inject a POJO or EJB data access object. Comment out the object not needed, see Example 5-7.

Example 5-7 Injecting dao into servlet

```
@WebServlet("/JPA21Servlet")
public class JPA21Servlet extends HttpServlet {
    // Uncomment for POJO dao
    // @Inject
    // PersonPojoDao personDao;

    // Comment for POJO dao
    @EJB
    PersonEJBDao personDao;
```

3. The `doPost` method reads the name and location parameters on the request to persist a new `Person` object and redirects to the `index.jsp` page. Example 5-8 shows a sample `doPost` method.

```

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String name = request.getParameter("name");
    String location = request.getParameter("location");
    Person p = new Person();
    p.setName(name);
    p.setLocation(location);
    personDao.persistPerson(p);

    response.sendRedirect("index.jsp");
}

```

Configuring server and testing JPA application

Before you can run the application, you need to configure Liberty server for JPA applications and create the database schema that holds persisted objects.

Preparing the server

To run the application, you have to enable the following features in the server configuration:

- ▶ jpa-2.1
- ▶ jsp-2.3
- ▶ cdi-1.2 - for Pojo example
- ▶ ejbLite-3.2 - for EJB example

You need to add features to the server configuration by using the following steps:

1. In the Servers view, expand the WebSphere Liberty server and double-click **Server Configuration**.
2. Select **Feature Manager** and in the Feature Manager section click **Add**, as shown in Figure 5-15.

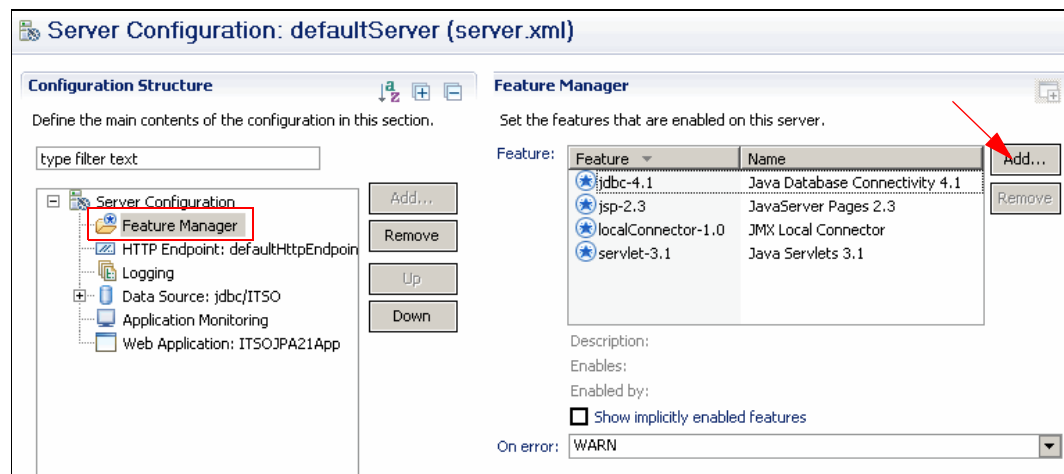


Figure 5-15 Adding features

3. In the *Select the features to enable* dialog, select **jpa-2.1**, **cdi-1.2**, and **ejbLite-3.2** features, and click **OK**. You also need to add the **jsp-2.3** feature, if it is not already present.
4. Save the server configuration.

Auto generating the schema for JPA entities

For development purposes, it is useful to quickly generate database schema. The schema can be based on your entities without a need to access database-specific tools or writing and running special schema generating scripts. This is possible in JPA 2.0, but you must use provider proprietary properties. JPA 2.1 introduces standard properties that allow you to generate schema, create scripts, and load data at start.

To configure automatic schema generation, use the following steps:

1. Open the `persistence.xml` file and select the **Schema Generation** tab.
2. In the Database action field, select **Create**. If you often change schema, you can select **Drop and Create**, but remember that all data is lost during redeployment.

If, in addition, you want to create scripts for later usage, select the required option in the Scripts generation field and provide file names in the Scripts target field, as shown in Figure 5-16. Scripts are generated in the server directory (`WLP_HOME/usr/servers/serverName`).

The screenshot shows the 'Schema Generation' configuration window. The 'Database action' dropdown is set to 'Create'. The 'Scripts generation' dropdown is set to 'Drop and Create'. The 'Scripts create target' field contains 'schemaCreate.ddl' and the 'Scripts drop target' field contains 'schemaDrop.ddl'. The 'Create database schemas (False)' checkbox is checked. The 'Database product name', 'Database major version', 'Database minor version', 'Create script source', and 'Drop script source' fields are empty.

Figure 5-16 Schema generation options

Testing application

Finally, test the application by visiting the following website:

<http://localhost:9080/ITS0JPA21App/index.jsp>

You should see output similar to Figure 5-17 on page 131. The table is empty on the first run because the database contains no records.

Hello, Liberty Developer from JPA.

These are our previous visitors:

Name	Location
Mary	Toronto
Thomas	Detroit
Pablo	Madrid
Kelly	Seattle

Name:

Location:

Figure 5-17 Output from the JPA application

5.1.3 Data access with noSQL database

NoSQL databases are becoming more popular. They are especially preferred for storing large amounts of unstructured data. Although they lack traditional database features (such as transaction management and defined schemas), their simple development and ability to scale horizontally make it a preferred choice for some applications.

WebSphere Liberty profile currently provides support for easy development with MongoDB and CouchDB.

Data access with MongoDB

The Liberty profile developer tools can facilitate the development of data access applications using MongoDB by simplifying the setup of required resources. Use the following sections and procedures to create an example application that stores, retrieves, and deletes data from a MongoDB database.

Installing MongoDB

For this example, you need a running instance of MongoDB. For more information about downloading, installing, and running MongoDB, go to the following website:

<http://www.mongodb.org>

Setting up the MongoDB Java driver

To set up the MongoDB Java driver, complete the following steps:

1. Download the MongoDB Java driver from the MongoDB website. Ensure that you download Version 2.10.1 or later. Save the Java driver JAR file in the location on disk where you want the Liberty profile server to access the driver.
2. Create a shared library. Select the **Server configuration**, click **Add**, and select **Shared Library**. In the details window for the shared library, enter the value `mongoLibrary` in the ID field. Click **Add** next to the Fileset reference field to add a file set to the shared library. In the Base directory field, enter the directory where you saved the MongoDB Java driver JAR file. In the Includes pattern field, enter the Java driver JAR file name. For Version 2.13.1, the file name is `mongo-java-driver-2.13.1.jar`.

Caution: At the time of writing this book, the Mongo 3.x driver was not working in Liberty.

Creating an application

To test the MongoDB access, you create a web application with two parts. A JSP file is used to display an input form, and a servlet is used to interact with the MongoDB instance and return results. Complete the following steps to write an application that uses MongoDB:

1. Begin by creating a new dynamic web project **ITS0MongoApp**.
2. Create a JSP file named `mongoInput.jsp` in the **ITS0MongoApp** project by using the New JSP File wizard. On the template selection page, choose **New JSP File**.
3. In the JSP file, add a form that has an input field named `item` and an input field named `price`. The action attribute for the form should be `MongoServlet` and the method attribute should be `POST`. Add three submit buttons to the form, labeled `Add to Catalog`, `Find in Catalog`, and `Delete from Catalog`. Figure 5-18 shows the completed `mongoInput.jsp` file.

```
<body>
  <form action="MongoServlet" method="POST">
    Item: <input type="text" name="item" />
    <BR> Price: <input type="text" name="price" />
    <BR> <input type="submit" name="add" value="Add to Catalog" />
    <input type="submit" name="find" value="Find in Catalog" />
    <input type="submit" name="delete" value="Delete from Catalog" />
  </form>
</body>
```

Figure 5-18 Example `mongoInput.jsp` file

4. Create a servlet named `MongoServlet` in the **ITS0MongoApp** project by using the Create Servlet wizard. In the servlet, access is needed to the APIs that are contained in the MongoDB Java driver JAR file. Add this JAR file to the build path for the **ITS0MongoApp** project by right-clicking the **project** and selecting **Properties**. Select **Java Build Path**, and then click the **Libraries** tab. Then, proceed with one of the following two options:
 - If your MongoDB Java driver JAR file is in your Eclipse workspace, click **Add** to add it to the build path.
 - Otherwise, click **Add External Jars** and navigate to the location of the JAR file. When you add the JAR file to the build path, click **OK** to update the project.
5. In the servlet, add a field of type `com.mongodb.DB` named `mongoDB`. Add the annotation `@Resource(name = "jndi/mongoDB")` to the field so that the Liberty profile server injects the DB instance automatically.
6. All of the database access takes place in the `doPost` method. In the `doGet` method, add the code in Example 5-9 to forward the request to the `mongoInput.jsp` file.

Example 5-9 Code to forward the request

```
RequestDispatcher rd = request.getRequestDispatcher("mongoInput.jsp");
rd.forward(request, response);
```

7. In the `doPost` method, obtain a `DBCollection` object by calling the `getCollection` method on the `mongoDB` field. The method parameter identifies the collection name. For this example, use the name `catalog`.

8. Also, in the **doPost** method, create a `com.mongodb.BasicDBObject` instance named `DBObject`. Store the request parameters for `item` and `price` in the `DBObject` by using the **append** method. Continue with the following configuration options:
 - If the request parameter `add` is not null, call **insert** on the `DBCollection` object, passing in `DBObject`.
 - If the request parameter `find` is not null, call **find** on the `DBCollection` object, passing in `DBObject`. The `find` method returns a `com.mongodb.DBCursor` object. Use the `size` method on `DBCursor` to determine the number of results.
 - If the request parameter `delete` is not null, run the **remove** method on the collection object, again passing in the `DBObject`. The **remove** method returns an object of type `com.mongodb.WriteResult`. You can use the **getN** method on the `WriteResult` object to determine how many results were deleted from the database.
9. Now call the **find** method on the `DBCollection` object without passing in any parameters. This returns a `DBCursor` object that contains all entries in the `catalog` collection. Iterate through the results using the **hasNext** and **next** methods on `DBCursor`. The **next** method returns a `DBObject` instance that you can use to obtain the data.

10. Finally, obtain a `RequestDispatcher` instance and call the `include` method to include the contents of the `mongoInput.jsp` file. The completed `doPost` method is shown in Figure 5-19.

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    PrintWriter writer = response.getWriter();
    writer.println("<BODY>");

    DBCollection collection = mongoDB.getCollection("catalog");

    BasicDBObject dbObject = new BasicDBObject();
    dbObject.append("Item", request.getParameter("item"));
    dbObject.append("price", request.getParameter("price"));

    if (request.getParameter("add") != null) {
        collection.insert(dbObject);
        writer.println("Successfully added new item.");
    } else if (request.getParameter("find") != null) {
        DBCursor cursor = collection.find(dbObject);
        if (cursor.hasNext()) {
            writer.println("<P>The item is in the catalog " + cursor.size()
                + " time" + (cursor.size() == 1 ? "" : "s"));
        } else {
            writer.println("The item was not found in the database");
        }
    } else if (request.getParameter("delete") != null) {
        WriteResult result = collection.remove(dbObject);
        String results = result.getN() == 1 ? " result was "
            : " results were ";
        writer.println(result.getN() + results + "deleted.");
    }

    writer.println("<P>All Database Entries");
    writer.println("<TABLE border=\"5\"><TH>Item</TH><TH>Price</TH>");
    DBCursor cursor = collection.find();
    while (cursor.hasNext()) {
        DBObject props = cursor.next();
        writer.println("<TR>");
        writer.println("<TD>" + props.get("Item") + "</TD>");
        writer.println("<TD>" + props.get("price") + "</TD>");
        writer.println("</TR>");
    }
    writer.println("</TABLE>");

    RequestDispatcher rd = request.getRequestDispatcher("mongoInput.jsp");
    rd.include(request, response);

    writer.println("</BODY>");
}
```

Figure 5-19 Example `doPost` method

Configuring MongoDB in the server configuration

To configure MongoDB in the server configuration, complete the following steps:

1. Create a Mongo instance in the server configuration:
 - a. Select the **Server configuration**, click **Add**, and then select **Mongo**. Accept the prompt to add the Mongo-2.0 Integration feature.
 - b. The details window contains several different options for configuring the Mongo instance, but you can keep all of the defaults except for the ID and MongoDB Java driver library reference. In the ID field, enter **mongo**. In the Java driver library reference field, use the drop-down menu to select **mongoLibrary**.
2. Create a MongoDB DB in the configuration:
 - a. Select the **server configuration**, click **Add**, and then select **MongoDB DB**.

- b. In the JNDI name field, enter the value `jndi/mongoDB`.
- c. In the database name field, enter `exampleDB`.
- d. Use the Mongo instance reference drop-down menu to select `mongo`.

The configuration is visible in `server.xml` in Example 5-10.

Example 5-10 Mongo db configuration in server.xml

```
<library id="mongoLibrary">
  <fileset dir="{shared.resource.dir}/mongo"
includes="mongo-java-driver-2.13.1.jar"></fileset>
</library>
<mongo id="mongo" libraryRef="mongoLibrary"></mongo>
<mongoDB jndiName="jndi/mongoDB" mongoRef="mongo"
databaseName="exampleDB"></mongoDB>
```

For more complex configurations visit the following page:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_mongodb_create.html?lang=en

Enabling the MongoDB feature

Mongo features should be already enabled in the `server.xml` as shown in Example 5-11.

Example 5-11 Mongo feature in server.xml

```
</featureManager>
  <feature>mongodb-2.0</feature>
</featureManager>
```

If the feature is not added, select the **Feature Manager** in the server configuration. Click **Add**, and select the `mongodb-2.0` feature.

Deploying the Application

Use the following steps to deploy applications:

1. Applications need access to the library you defined for the Mongo Java driver. You cannot just drop it to the `dropins` folder, you have to configure it in the `server.xml` file (either using tools in Eclipse or manually). Here you will use Eclipse.
2. Select **Liberty server**, right-click and select **Add and Remove**, select **ITSOMongoApp**, click **Add**, and then click **Finish**.
3. Open **Server configuration**, select **Web Application: ITSOMongoApp** and click **Add**.
4. Select **Classloader** and click **OK**.
5. In the Classloader Details window, click the **Browse** button next to the **Shared library references** field. Select `mongoLibrary` from the list of available IDs and click **OK**.

The following fragment of `server.xml` shows some application configuration, see Example 5-12.

Example 5-12 Application configuration in server.xml

```
<webApplication id="ITSOMongoApp" location="ITSOMongoApp.war" name="ITSOMongoApp">
  <classloader commonLibraryRef="mongoLibrary"></classloader>
</webApplication>
```

Testing the application

Test the application by visiting the following website:

<http://localhost:9080/ITSOMongoApp/MongoServlet>

Result: You should be able to use the web page to add, find, and remove items from the database. For the find operation to succeed, both input fields must match the corresponding fields in a database entry.

Data access with CouchDB

The Liberty profile developer tools can facilitate the development of data access applications using CouchDB by simplifying the setup of required resources. Use the following sections and procedures to create an example application that stores, retrieves, and deletes data from a CouchDB database.

Installing CouchDB

For this example, you need a running instance of CouchDB. For more information about downloading, installing, and running CouchDB, go to the following website:

<http://docs.couchdb.org/en/1.6.1>

Setting up the CouchDB Java connector

To set up the CouchDB Java connector, complete the following steps:

1. Download the CouchDB Java connector libraries. You must use Version 1.4.1 or later of the ektorp Java connector. The CouchDB Java connector is a set of files defined by maven dependency. Use the Maven plug-in to obtain it using the dependency shown in Example 5-13.

Example 5-13 Connector dependencies

```
<dependency>
  <groupId>org.ektorp</groupId>
  <artifactId>org.ektorp</artifactId>
  <version>1.4.1</version>
</dependency>
```

You can also download JAR files separately by using the list in Example 5-14 on page 136.

Example 5-14 JAR files needed by the connector

```
org.ektorp-1.4.1.jar
commons-codec-1.6.jar
commons-io-2.0.1.jar
commons-logging-1.1.1.jar
httpclient-4.2.5.jar
httpclient-cache-4.2.5.jar
httpcore-4.2.4.jar
jackson-annotations-2.2.2.jar
jackson-core-2.2.2.jar
jackson-databind-2.2.2.jar
slf4j-api-1.6.4.jar
slf4j-simple-1.6.4.jar
```

Save the Java connector JAR files in the location on disk where you want the Liberty profile server to access the driver.

2. Create a shared library. Select the **Server configuration**, click **Add**, and select **Shared Library**. In the details window for the shared library, enter the value `couchLibrary` in the ID field. Click **Add** next to the Fileset reference field to add a file set to the shared library. In the Base directory field, enter the directory where you saved the CouchDB Java connector JAR file. In the Includes pattern field, enter `*.jar` to include all the JAR files from the folder.

Creating an application

To test the CouchDB access, you create a web application with two parts. A JSP file is used to display an input form, and a servlet is used to interact with the CouchDB instance and return results. Complete the following steps to create an application that uses CouchDB:

1. Begin by creating a new dynamic web project **ITSOCouchApp**.
2. Create a JSP file named `couchInput.jsp` in the **ITSOCouchApp** project by using the New JSP File wizard. On the template selection page, choose **New JSP File**.
3. In the JSP file, add a form that has an input field named `item` and an input field named `price`. The action attribute for the form should be `CouchServlet` and the method attribute should be `POST`. Add three submit buttons to the form, labeled `Add to Catalog`, `Find in Catalog`, and `Delete from Catalog`. Example 5-15 shows the completed `couchInput.jsp` file.

Example 5-15 couchInput.jsp fragment

```
<body>
  <form action="CouchServlet" method="post">
    ID: <input type="text" name="id"><BR>
    Item: <input type="text" name="item"><BR>
    Price: <input type="text" name="price"><BR>
    <input type="submit" name="add" value="Add to Catalog">
    <input type="submit" name="find" value="Find in Catalog">
    <input type="submit" name="delete" value="Delete fromCatalog">
  </form>
</body>
```

4. Create a servlet named `CouchServlet` in the **ITSOCouchApp** project by using the Create Servlet wizard. In the servlet, access is needed to the APIs that are contained in the CouchDB Java connector JAR files. Add these JAR files to the build path for the **ITSOCouchApp** project by right-clicking the **project** and selecting **Properties**. Select **Java Build Path**, and then click the **Libraries** tab. Then, proceed with one of the following two options:
 - If your CouchDB Java connector JAR files are in your Eclipse workspace, click **Add** to add it to the build path.
 - Otherwise, click **Add External Jars** and navigate to the location of the JAR files. When you add the JAR files to the build path, click **OK** to update the project.
5. In the servlet, add a field of type `org.ektorp.CouchDbInstance` named `couchDB`. Add the annotation `@Resource(name = "couchdb/connector")` to the field so that the Liberty profile server injects the DB instance automatically.
6. All of the database access takes place in the `doPost` method. In the `doGet` method, add the code in Example 5-16 to forward the request to the `couchInput.jsp` file.

Example 5-16 Code to forward the request

```
RequestDispatcher rd = request.getRequestDispatcher("couchInput.jsp");
```

```
rd.forward(request, response);
```

7. In the **doPost** method, obtain a CouchDbConnector object by calling the **createConnector** method on the couchDB field. The method parameters identify the database name and database creation, if not existing. For this example, use the name `examp1edb`.
8. Also, in the **doPost** method, create a HashMap instance named `doc`. Store the request parameters for `id`, `item`, and `price` in the `doc`. Put the key value as `_id`. Continue with the following configuration options:
 - If the request parameter `add` is not null, call **create** on the CouchDbConnector object, passing in `doc` object.
 - If the request parameter `find` is not null, call **find** on the CouchDbConnector object, passing in result class type and object key value. The `find` method returns a Map object that you put into the database.
 - If the request parameter `delete` is not null, first look for the object to delete by using the **find** method. If found, run the **delete** method passing in the found object.
9. Now call the **queryView** method on the CouchDbConnector object passing ViewQuery object. This returns a List that contains all entries in the `examp1edb` database. Iterate through the results list to display the data.
10. Finally, obtain a RequestDispatcher instance and call the **include** method to include the contents of the `couchInput.jsp` file. The completed **doPost** method is shown in Example 5-17.

Example 5-17 doPost method implementation

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    PrintWriter writer = response.getWriter();
    writer.println("<BODY>");
    String dbname = "examp1edb";
    try {
        //creates a database with the specified name
        CouchDbConnector dbc = couchDB.createConnector(dbname, true);

        //create a simple doc to place into your new database
        Map<String, Object> doc = new HashMap<String, Object>();
        doc.put("_id", request.getParameter("id"));
        doc.put("item", request.getParameter("item"));
        doc.put("price", request.getParameter("price"));

        if (request.getParameter("add") != null) {
            dbc.create(doc);
            writer.println("Added a simple doc!");
        } else if (request.getParameter("find") != null) {
            Map result = dbc.find(Map.class, request.getParameter("id"));
            if (result != null) {
                writer.println("<P>The item is in the catalog: " + result);
            } else {
                writer.println("The item was not found in the database");
            }
        } else if (request.getParameter("delete") != null) {
            Map result = dbc.find(Map.class, request.getParameter("id"));
            if (result != null) {
```

```

        writer.println("<P>The item is in the catalog - deleting.... " +
result);
        dbc.delete(result);
    } else {
        writer.println("The item was not found in the database");
    }
}
writer.println("<P>All Database Entries");
writer.println("<TABLE
border=\"5\"><TH>ID</TH><TH>Item</TH><TH>Price</TH>");
ViewQuery q = new ViewQuery().allDocs().includeDocs(true);
List<Map> bulkLoaded = dbc.queryView(q, Map.class);

for (Map map : bulkLoaded) {
    writer.println("<TR>");
    writer.println("<TD>" + map.get("_id") + "</TD>");
    writer.println("<TD>" + map.get("item") + "</TD>");
    writer.println("<TD>" + map.get("price") + "</TD>");
    writer.println("</TR>");
}
writer.println("</TABLE>");
}
catch(Exception e){
    e.printStackTrace();
}
RequestDispatcher rd = request.getRequestDispatcher("couchInput.jsp");
rd.include(request, response);
writer.println("</BODY>");
}
}

```

Configuring CouchDB in the server configuration

To configure CouchDB in the server configuration, complete the following process.

Create a Couch instance in the server configuration:

- ▶ Select the Server configuration, click **Add**, and then select **Couch**. Accept when prompted to add the couchdb-1.0 Integration feature.
- ▶ The details window contains several different options for configuring the Couch instance. Provide the following options, as shown in Figure 5-20 on page 140:
 - ID: couchdb
 - Library reference: couchLibrary
 - CouchDB user id: admin
 - Password: mysecretpassword
 - JNDI name: couchdb/connector
 - URL: http://localhost:5984



Figure 5-20 CouchDB options

The configuration is visible in `server.xml` in Example 5-18.

Example 5-18 Couch db configuration in `server.xml`

```
<library id="couchLibrary">
  <fileset dir="{shared.resource.dir}/crouch" includes="*.jar"/>
</library>
<couchdb id="couchdb" jndiName="couchdb/connector"
  libraryRef="couchLibrary" url="http://localhost:5984"
  username="admin" password="mysecretpassword"/>
```

Enabling the CouchDB feature

Couch features should be already enabled in the `server.xml` as shown in Example 5-19.

Example 5-19 Couch feature in `server.xml`

```
</featureManager>
  <feature>couchdb-1.0</feature>
</featureManager>
```

If the CouchDB feature is not added, select the **Feature Manager** in the server configuration. Click **Add**, and select the **couchdb-1.0** feature.

Deploying Application

Use the following steps to deploy the application:

1. Applications need access to the library you defined for the CouchDB Java connector. You cannot just drop it into the dropins folder. You have to configure it in the `server.xml`, either by using tools in Eclipse or manually. In this example, use Eclipse.
2. Select **Liberty server**, right-click and select **Add and Remove**. Select **ITSOCouchApp**, click **Add >**, and click **Finish**.
3. Open Server configuration, select **Web Application: ITSOCouchApp** and click **Add**.
4. Select **Classloader** and click **OK**.
5. In the Classloader Details window, click the **Browse** button next to the Shared library references field. Select **couchLibrary** from the list of available IDs and click **OK**.

The following fragment of `server.xml` shows some of the application configuration in Example 5-20.

Example 5-20 Application configuration in `server.xml`

```
<webApplication id="ITSOcrouchApp" location="ITSOcrouchApp.war"
name="ITSOcrouchApp">
  <classloader commonLibraryRef="couchLibrary"></classloader>
</webApplication>
```

Testing the application

Test the application by visiting the following website:

`http://localhost:9080/ITSOcouchApp/CouchServlet`

Result: You should be able to use the web page to add, find, and remove items from the database. For the find and delete operations to succeed, you must provide an ID field.

Not authorized: If you receive the 401:Unauthorized error from the call, configure the admin user in CouchDB. In the `C:\CouchDB\etc\couchdb\local.ini` file at the end (in the `[admins]` section entry), uncomment the entry `admin = mysecretpassword`.

5.2 Developing Enterprise JavaBeans applications

The Enterprise JavaBeans (EJB) specification describes a component model for business logic in server applications. WebSphere Liberty Profile Server supports the EJB 3.2, EJB 3.2 Lite specification, and message-driven beans (MDBs).

5.2.1 What's new in developing EJB applications for Liberty profile

WebSphere Liberty profile now supports full EJB 3.2 specification in addition to EJB Lite. The latest available features include those noted in the following list:

- ▶ Local and remote EJB
- ▶ Support for persistent and non-persistent timers
- ▶ Support for asynchronous session bean invocations

5.2.2 Developing applications using local EJB

The Liberty profile developer tools facilitate the development of EJB applications by providing wizards that help generate portions of the EJB code. Use the following sections and procedures to create and test a simple stateless local session bean that uses `ejblite-3.2`. Since Java EE now allows embedding EJBs in the web modules, this approach is shown in this section. WebSphere Liberty also fully supports creating an EAR project with separate web and EJB modules.

Creating a Dynamic Web Project

To create a project, complete the following steps:

1. Begin by creating a new project to hold the EJB application. Click **File** → **New** → **Dynamic Web Project**. On the first page of the New Project wizard, enter `ITSOEJBLoca1` in the Project name field.

2. Make sure that WebSphere Application Server V8.5 Liberty Profile is selected for the Target run time. Also, ensure that the **Add project to an EAR** check box is cleared.
3. Click **Finish** to complete the wizard. A new project named `ITSOEJBLocal` is created after you exit the wizard.

Creating the EJB

After creating the project, complete the following steps to create the EJB:

1. From the menu, select **File** → **New** → **Session Bean (EJB 3.x)**, which starts the Create EJB 3.x Session Bean wizard.
2. In the first window of the wizard, ensure that `ITSOEJBLocal` is selected in the Project field, enter `com.ibm.itso.ejblocal` in the Java package field, and `Hello` in the Class name field. Ensure that `Stateless` is specified for the State type field, and default **No-interface View** is selected. When you select the No-interface View, all public, non-static methods of the bean are available to clients. If you do not want that availability, select **Local** and define the interface with only required methods. The EJB creation wizard is shown in Figure 5-21.

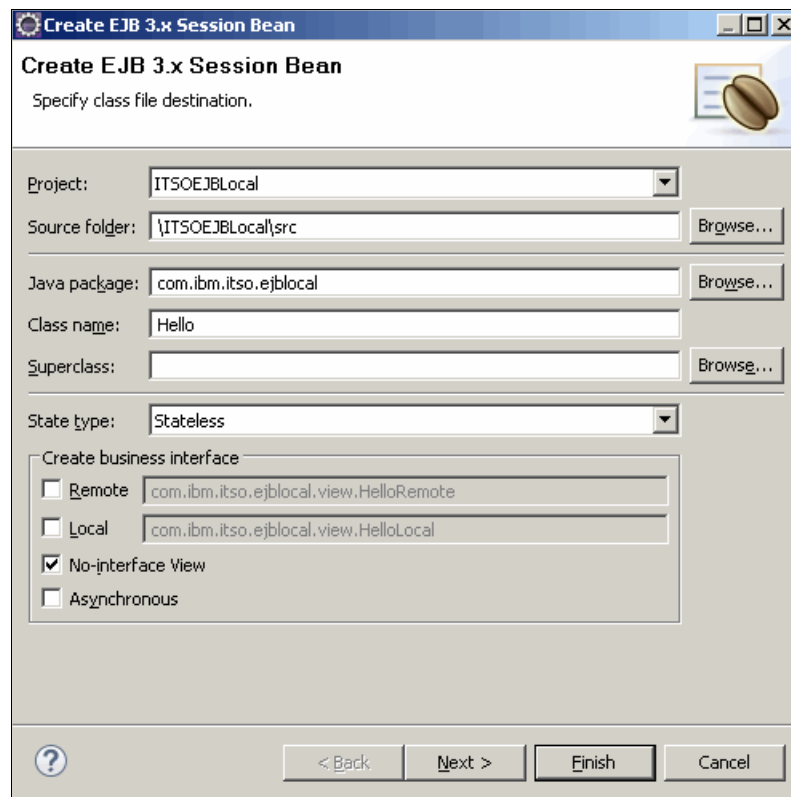


Figure 5-21 Creating EJB

3. Click **Next** to advance to the next window, and then click **Finish**. The wizard now creates a class named `Hello`. Notice that the class is annotated with `@Stateless` and `@LocalBean`.
4. Now, complete the EJB implementation by adding a method that is named `hello` that returns the string `Hello` from `local` bean. The completed class is shown in Example 5-21.

Example 5-21 Hello bean class

```
@Stateless
@LocalBean
public class Hello {
```



```

    public String hello() {
        return "Hello from local bean";
    }
}

```

Creating a test client

A servlet is used to test the created bean. To create a servlet, complete the following steps:

1. Using the Create New Servlet wizard, create a servlet in the `ITS0EJBLocal` project, in the package `com.ibm.itso.ejbLocal` named `HelloServlet`. In the generated class, add a field named `helloBean` of type `Hello`. Annotate the field with `@EJB`.
2. In the servlet's `doGet` method, print the results of calling the `hello` method on `helloBean`. The completed servlet is shown in Example 5-22.

Example 5-22 HelloServlet class

```

@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @EJB
    private Hello helloBean;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println(helloBean.hello());
    }
}

```

Enabling the EJB feature and testing

You can test `HelloServlet` directly from the workspace:

- ▶ Select the **HelloServlet** file, right-click, and select **Run as** → **Run on Server**
- ▶ Browser in Eclipse is automatically opened pointing to the following website:
`http://localhost:9080/ITS0EJBLocal/HelloServlet`

Result: You should see the output, `Hello from the local bean`, in your web browser. This shows that the EJB was automatically injected into the `helloBean` field and that the servlet was able to successfully start the method `hello`.

By default, WebSphere Liberty server automatically loads the required features if they are available in the runtime. You should see, in the console, output messages similar to those in Example 5-23 on page 143 where the server is automatically adding `jndi-1.0` and `ejbLite-3.2`; you can also notice default binding for the `Hello` bean.

Example 5-23 Automatically enabling features

```

[AUDIT ] CWWKG0016I: Starting server configuration update.
[INFO  ] CWWKF0007I: Feature update started.
[AUDIT ] CWWKG0017I: The server configuration was successfully updated in 0.856
seconds.

```

```
[AUDIT ] CWWKF0012I: The server installed the following features: [jndi-1.0,
ejbLite-3.2].
[AUDIT ] CWWKF0008I: Feature update completed in 0.846 seconds.
[INFO ] CWWKZ0018I: Starting application ITS0EJBLocal.
[INFO ] CNTR4000I: The ITS0EJBLocal.war EJB module in the ITS0EJBLocal
application is starting.
[INFO ] CNTR0167I: The server is binding the com.ibm.itso.ejblocal.Hello
interface of the Hello enterprise bean in the ITS0EJBLocal.war module of the
ITS0EJBLocal application. The binding location is:
java:global/ITS0EJBLocal/Hello!com.ibm.itso.ejblocal.Hello
[INFO ] CNTR4001I: The ITS0EJBLocal.war EJB module in the ITS0EJBLocal
application has started successfully.
[INFO ] SRVE0169I: Loading Web Module: ITS0EJBLocal.
[INFO ] SRVE0250I: Web Module ITS0EJBLocal has been bound to default_host.
[AUDIT ] CWWKT0016I: Web application available (default_host):
http://localhost:9080/ITS0EJBLocal/
[AUDIT ] CWWKZ0001I: Application ITS0EJBLocal started in 0.313 seconds.
[AUDIT ] CWWKG0016I: Starting server configuration update.
```

The required features can also be added by opening the server configuration and selecting **Feature Manager** in the server configuration. Then, click **Add**, and select **ejbLite-3.2** and **jndi-1.0**.

5.2.3 Developing applications using remote EJB

Liberty also supports remote EJB as required by the Java EE 7 full platform specification. In this chapter, you create a project that has remote EJB and two types of clients (separate web application and Java client application).

Creating an EJB project

To create a project, complete the following steps:

1. Begin by creating a new project to hold the EJB application. Click **File** → **New** → **EJB Project**. On the first page of the New EJB Project wizard, enter ITS0RemoteEJB in the Project name field, ensure that **Add project to an EAR** is checked and provides ITS0Remote as the name for the EAR. Click **Next**. Then, again click **Next**.
2. In the **Configure EJB module settings** window, ensure that **Create an EJB Client JAR** is selected, then click **Finish**.

The EJB client JAR file is used by remote clients to find required class files.

Creating the EJB

After creating the project, complete the following steps to create the EJB:

1. From the menu select **File** → **New** → **Session Bean (EJB 3.x)**, which starts the Create EJB 3.x Session Bean wizard.

- In the first window of the wizard, ensure that **ITSORemoteEJB** is selected in the Project field. Enter `com.ibm.itso.ejbRemote` in the Java package field and `HelloRemoteEJB` in the Class name field. Ensure that `Stateless` is specified for the State type field, and change the business interface option from `No-interface View` to `Remote` (shown in Figure 5-22).

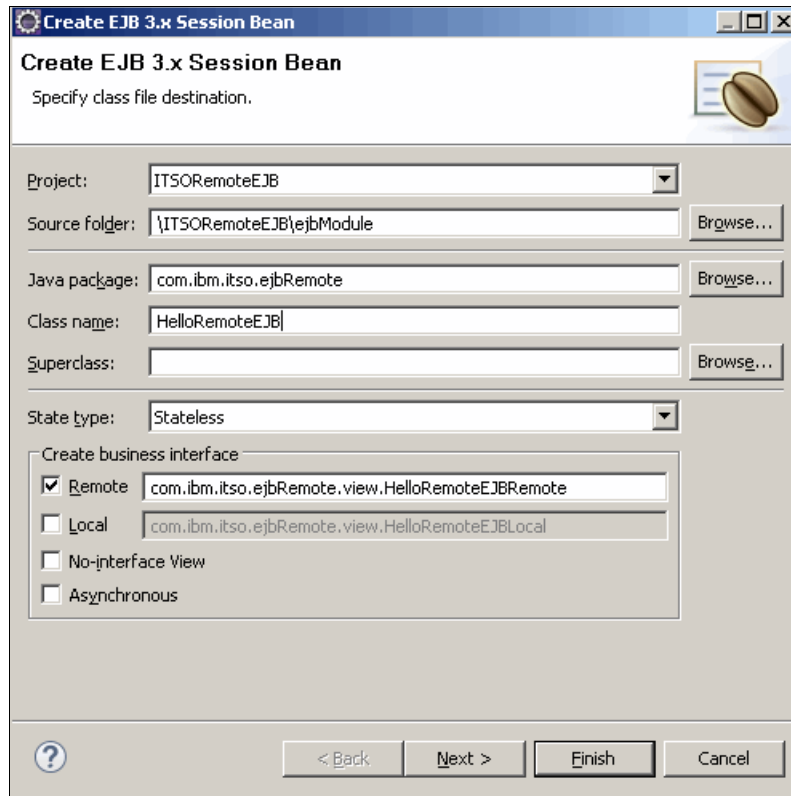


Figure 5-22 Creating remote EJB

- Click **Next** to advance to the next window, and then click **Finish**. The wizard now creates a class named `HelloRemoteEJB` and also an interface called `HelloRemoteEJBRemote` in the `ITSORemoteEJBClient` project. Notice that the class is annotated with `@Stateless` and `@Remote`.
- Now, complete the EJB implementation by adding a method that is named `hello` that returns the string `Hello` from remote bean. The completed class is shown in Example 5-24.

Example 5-24 `HelloRemoteEJB` class

```

@Stateless
@Remote>HelloRemoteEJBRemote.class)
public class HelloRemoteEJB implements HelloRemoteEJBRemote {
    public String hello() {
        return "Hello from remote bean!";
    }
}

```

- Promote the `hello()` method to the remote interface. In the Outline view, click `hello()` method, right-click, and select **Java EE Tools** → **Promote Methods**. Accept the defaults and click **OK**.

Enabling the feature and deploying to the server

To enable `ejbRemote-3.2` in the server configuration, use the following steps:

1. In the Servers view, expand Liberty Server and double-click **Server Configuration [server.xml]**.
2. Select **Feature Manager** and in the Feature Manager section of the window, click the **Add** button and add the `ejbRemote-3.2` feature.
3. Save the changes.

Now, deploy `ITSORemote` application to the server. To do so, in the Servers view right-click Liberty Server, then click **Add and Remove**, and add `ITSORemote` application.

Your remote EJB application is ready for testing.

Creating a web test client

A separate web project is created to show the remote access to the EJB. Complete the following steps to create a web test client:

1. Begin by creating a new web project to hold the client application. Click **File** → **New** → **Dynamic Web Project**. On the first page of the New Project wizard, enter `ITSORemoteWebClient` in the Project name field.
2. Make sure that **WebSphere Application Server V8.5 Liberty Profile** is selected for the Target run time. Also, ensure that the **Add project to an EAR** check box is cleared.
3. Click **Finish** to complete the wizard. A new project named `ITSORemoteWebClient` is created after you exit the wizard.
4. Right click the `ITSORemoteWebClient` project and select **Properties**. In the project properties select **Deployment Assembly**.

- Click the **Add** button, select **Project** and then **ITSORemoteEJBClient**. Click **Finish**. Deployment Assembly properties should be updated with the **ITSORemoteEJBClient.jar** as shown in Figure 5-23.

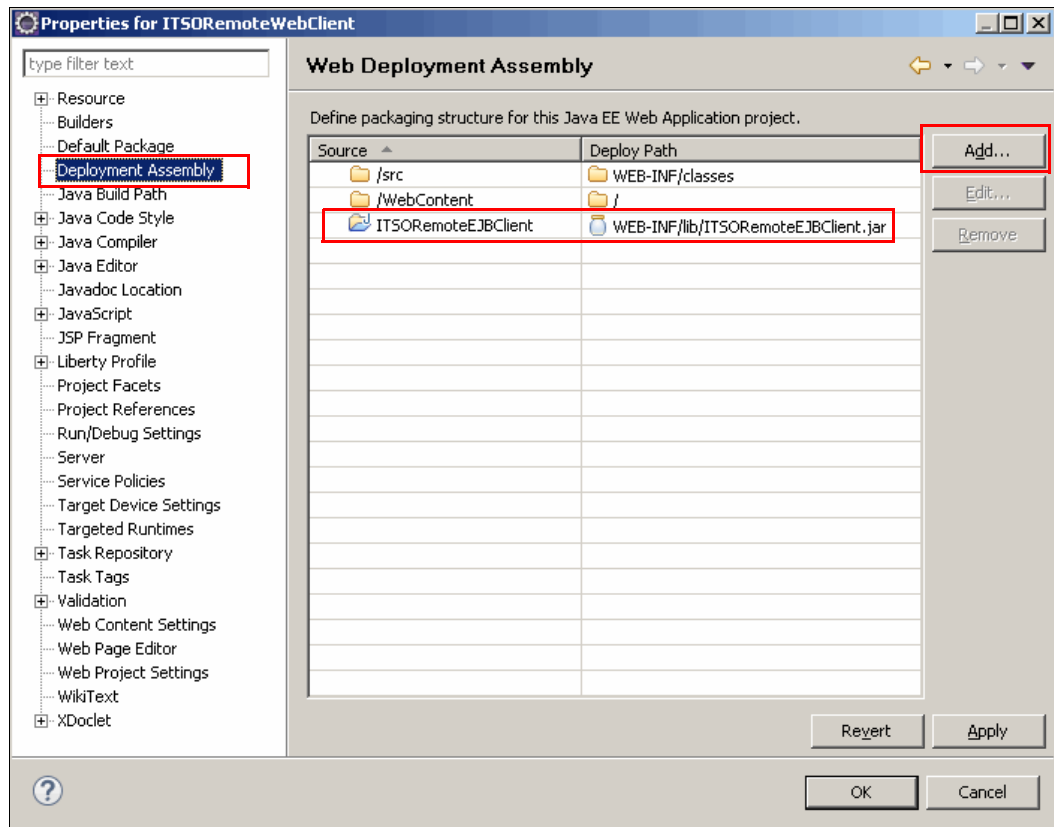


Figure 5-23 Deployment assembly properties

- Click **OK** to close the properties window and save the changes.
- Using the Create New Servlet wizard, create a servlet in the **ITSORemoteWebClient** project, in the package **com.ibm.itso.ejbRemote** named **HelloRemoteServlet**. In the generated class, add a field named **helloRemoteBean** of type **HelloRemoteEJBRemote**. Annotate the field with **@EJB**.
- In the servlet's **doGet** method, print the results of calling the **hello** method on **helloBean**. The completed servlet is shown in Example 5-25.

Example 5-25 HelloServlet class

```
@WebServlet("/HelloRemoteServlet")
public class HelloRemoteServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB(lookup="java:global/ITSORemote/ITSORemoteEJB/HelloRemoteEJB!com.ibm.itso.e
    jbRemote.view>HelloRemoteEJBRemote")
    private HelloRemoteEJBRemote helloRemoteBean;

    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
```

```
        out.println(helloRemoteBean.hello());
    }
}
```

JNDI name for remote EJB:

For beans in the same Liberty server, use:

```
java:global/ApplicationName/ModuleName/EJBName!full.package.remote.interface.Name
```

For beans in a different Liberty server, use:

```
corbaname:differentHostName:2809#ejb/global/ApplicationName/ModuleName/EJBName!full.package.remote.interface.Name
```

You can now test directly from the workspace:

1. Select the HelloRemoteServlet file, right-click, and select **Run as** → **Run on Server**
2. Browser is automatically opened pointing to
<http://localhost:9080/ITSORemoteWebClient/HelloRemoteServlet>

Result: You should see the output Hello from the remote bean in your web browser. This shows that the EJB was automatically injected into the helloRemoteBean field and that the servlet was able to successfully call the method `hello` in the remote bean.

Creating a Java application test client

WebSphere Liberty now supports running Java applications, in the application client container, that can access your components running on the Liberty server (for example remote EJB).

Using the client container is a two part process. First, you need to create an application. Then, configure and run that application using a client container.

Creating a Java application client

Use the following steps to create a Java application client:

1. Start by creating the new application client project. From the menu, select **File** → **New** → **Application Client Project**. On the first page of the New Project wizard, enter `ITSOHelloJavaClient` in the Project name field, type `ITSOHelloJavaClientEAR` in the EAR project name, and select **7.0** as the Application client module version. See Figure 5-24 on page 149. Then, click **Next** twice.

Application Client module
Create an Application Client project and add it to a new or existing Enterprise Application project

Project name: ITSOHelloJavaClient

Project location
 Use default location
Location: C:\workspaceResidency\ITSOHelloJavaClient

Target runtime
<None>

Application client module version
7.0

Configuration
<custom>

EAR membership
 Add project to an EAR
EAR project name: ITSOHelloJavaClientEAR

Working sets
 Add project to working sets
Working sets:

< Back Next > Finish Cancel

Figure 5-24 Creating an application client project

Attention: At the time of writing this book, it was not possible to select the Liberty Target run time while creating the Application client project. As a result, you need to later on add the Server Library to the project.

2. In the Configure Client module settings window, clear **Create default Main class**, and **check** to **Generate application-client.xml**. Click **Finish**.
3. Add the server run time to the project:
 - a. Right-click the `ITSOHelloJavaClient` project and select **Build Path** → **Configure Build Path**.

- b. Click **Add Library**. Select **Server Runtime**. Select your **WebSphere Liberty** runtime and click **Finish**. Click **OK**.
4. Add ITSORemoteEJBClient library to the application libraries:
 - a. Right-click **ITS0HelloJavaClientEAR** and select **Properties**.
 - b. Select **Deployment Assembly** in the left section of the Properties window, as shown in Figure 5-25 on page 150, and click **Add**.

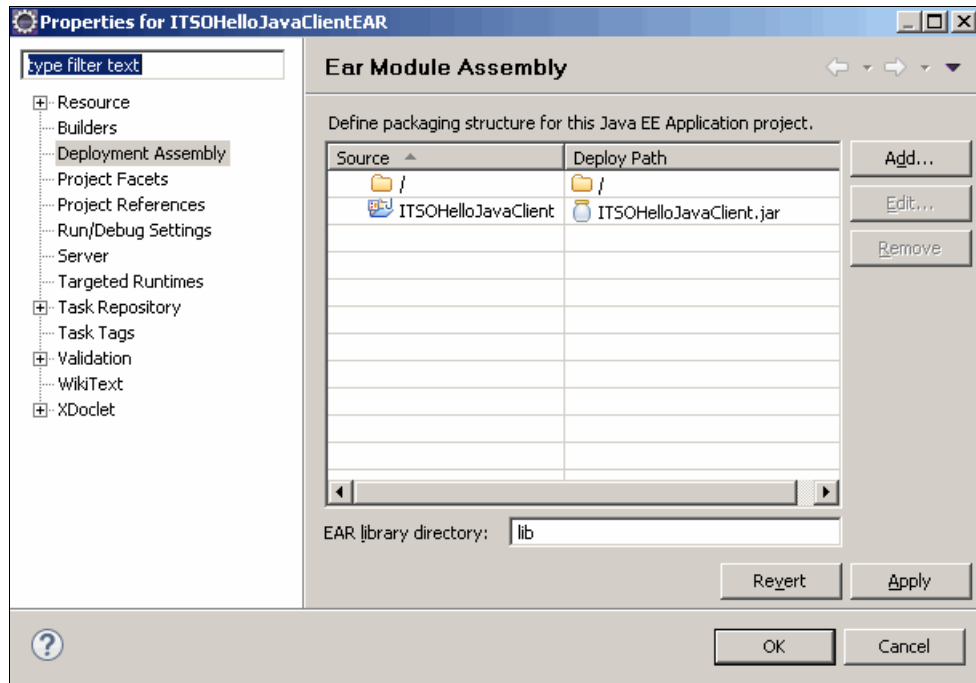


Figure 5-25 Configuring Deployment Assembly

- c. Select **Project** and click **Next**.
- d. Select **ITSORemoteEJBClient** and click **Finish**.
- e. Click **OK** to close Properties window.
5. Create the main application class by using the New Java Class wizard. Enter `com.ibm.itso.applient` as the Package name, `HelloClient` as Class name, and **check to Create the main method**. Click **Finish**.
6. Add the injected EJB as the static class field, and code the main method to call EJB. See the complete code in Example 5-26.

Example 5-26 Complete Hello client class

```
public class HelloClient {

    @EJB(lookup="corbaname::localhost:2809#ejb/global/ITSORemote/ITSORemoteEJB/HelloRemoteEJB!com.ibm.itso.ejbRemote.view>HelloRemoteEJBRemote")
    static HelloRemoteEJBRemote remoteBean;

    public static void main(String[] args) {
        System.out.println("Starting Liberty java client");
        System.out.println(remoteBean.hello());
    }
}
```


}

7. For the application to automatically run, it needs to have the Main-class attribute configured in the MANIFEST.MF file. Use the following steps for that process:
 - a. Expand **ITSORemoteEJBClient** → **appClientModule** → **META-INF** and double-click **MANIFEST.MF**.
 - b. Scroll down and beside Main-Class, click **Browse** and select **HelloClient** class, as shown in Figure 5-26 on page 151.

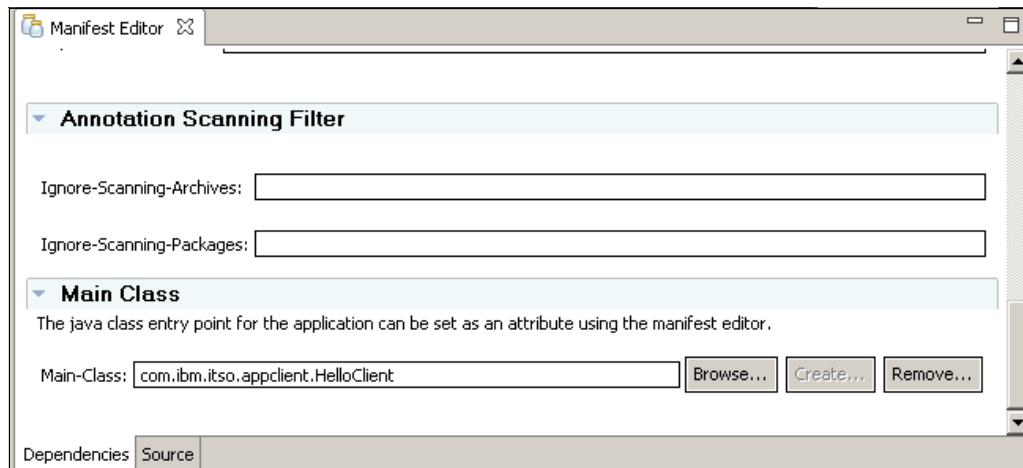


Figure 5-26 Defining Main-Class in the manifest file

8. Create the application deployment descriptor by right-clicking the **ITSOHelloJavaClientEAR** and select the **Java EE Tools** → **Generate Deployment Descriptor Stub**.
9. To export the application, right-click **ITSOHelloJavaClientEAR** and select **Export** → **EAR file**. Now, the **ITSOHelloJavaClientEAR** application is ready to be deployed in the application client container.

Running the application client: At the time of writing the book, it was not possible to run WebSphere Liberty application client directly from Eclipse. You need to export the application and configure application client container as described in the next section.

Configuring the Java application client container

To run the Java application client for WebSphere Liberty, you need to create and configure the container. To create the container, use the following steps:

1. Start by running the **client create clientName** command from the **WLP_HOME\bin** directory, providing **ITSOHelloClient** as the client name, as shown in Example 5-27.

Example 5-27 Creating client container

```
C:\IBM\WebSphere\wlp\bin>client create ITSOHelloClient
Client ITSOHelloClient created.
```

2. Observe that the **ITSOHelloClient** folder was created in the **C:\IBM\WebSphere\wlp\usr\clients** directory.
3. Copy the previously exported **ITSOHelloJavaClientEAR.ear** file to the **C:\IBM\WebSphere\wlp\usr\clients\ITSOHelloClient\apps** folder.

4. Edit the client configuration file, `client.xml`, in `C:\IBM\WebSphere\wlp\usr\clients\ITSOHelloClient` (See Example 5-28).

Example 5-28 Client configuration file, client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<client description="new client">
  <featureManager>
    <feature>javaeeClient-7.0</feature>
  </featureManager>
  <application id="ITSOHelloJavaClientEAR_ID" name="ITSOHelloJavaClientEAR"
type="ear" location="ITSOHelloJavaClientEAR.ear"/>
</client>
```

5. Run your application by using the client container. This container executes the `client run clientName` command, as shown in Example 5-29.

Example 5-29 Running application

```
C:\IBM\WebSphere\wlp\bin>client run ITSOHelloClient
```

6. Observe the output from the successful application execution, as shown in Example 5-30.

Example 5-30 Output from the running application

```
...
[AUDIT ] CWWKZ0001I: Application ITSOHelloJavaClientEAR started in 1.578
seconds.
...
[AUDIT ] CWWKF0035I: The client ITSOHelloClient is running.
Starting Liberty java client
Hello from remote bean!
[AUDIT ] CWWKZ0009I: The application ITSOHelloJavaClientEAR has stopped
successfully.
[AUDIT ] CWWKE0908I: The client ITSOHelloClient stopped after 8.216 seconds.
```

5.2.4 Developing applications using timers

Applications often need to perform some tasks at a specific time or repeatedly (such as generating daily reports). The timer service allows you to schedule notifications according to a calendar schedule, at a specific time, after a duration of time, or at timed intervals. It is available to all enterprise beans *except* stateful.

Timers can be either automatic (created by the `@Schedule` annotation), or programmatic (created explicitly by `TimerService` methods). Automatic timers are created by the container during the application start.

Timers can also be non-persistent or persistent. Non-persistent timers configuration is held only in JVM memory. In case of application or server restart, it needs to be re-created. Non-persistent timers run in every JVM of the cluster, where the application is deployed. Persistent timers configuration is stored in the database and any missed notifications, for example, due to a stopped application, will be reissued after the restart.

Using non-persistent timers

Use the following steps to create non-persistent timers:

1. Start by creating a new project that holds your timer class.

2. Create a new class with the code shown in Example 5-31 on page 153. Make sure to have **persistent=false** (as by default, timers are persistent). This timer runs on the tenth second every minute.

Example 5-31 Non-persistent timer bean

```
@Singleton
@Startup
public class TimerBean {
    @PostConstruct
    public void init() {
        System.out.println("TimerBean initialized");
    }
    @Schedule(second="10", minute="*", hour="*", persistent=false )
    public void generateReport() {
        System.out.println("Generating report...");
        // your logic here....
    }
}
```

3. Configure the server to run the application with a non-persistent timer:
 - a. In the **Servers** view, expand your Liberty server and double-click **Server configuration**.
 - b. Add the **ejbLite-3.2** feature, as shown in Figure 5-27.

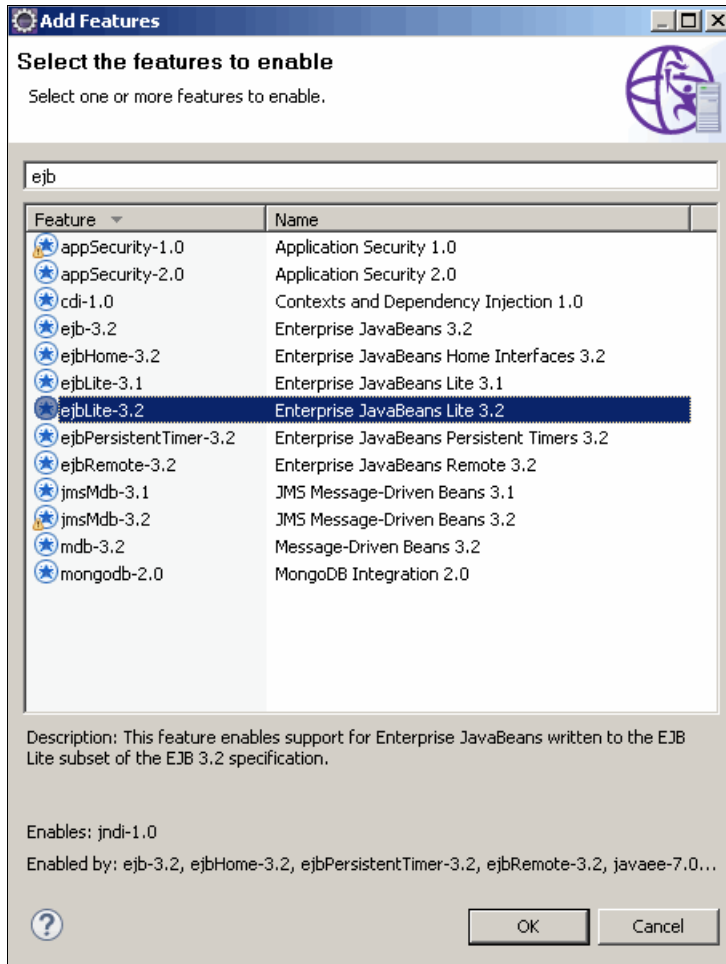


Figure 5-27 Adding the `ejbLite-3.2` feature

You can also enable the `ejbLite-3.2` feature manually, adding the snippet shown in Example 5-32 to the `server.xml`.

Example 5-32 `ejbLite` feature in `server.xml`

```
<featureManager>
  <feature>ejbLite-3.2</feature>
</featureManager>
```

- Now, deploy your application to the server. After a short time, you should see the output in the console (as shown in Example 5-33).

Example 5-33 Non-persistent timer running

```
[INFO ] CNTR4000I: The ITSOTimerApp.war EJB module in the ITSOTimerApp
application is starting.
[INFO ] CNTR0167I: The server is binding the com.ibm.itso.timers.TimerBean
interface of the TimerBean enterprise bean in the ITSOTimerApp.war module of
the ITSOTimerApp application. The binding location is:
java:global/ITSOTimerApp/TimerBean!com.ibm.itso.timers.TimerBean
[INFO ] CNTR0219I: The server created 0 persistent automatic timer or timers
and 1 non-persistent automatic timer or timers for the ITSOTimerApp.war module.
...
```

```
[INFO ] CNTR4001I: The ITSOTimerApp.war EJB module in the ITSOTimerApp
application has started successfully.
[INFO ] SRVE0169I: Loading Web Module: ITSOTimerApp.
[INFO ] SRVE0250I: Web Module ITSOTimerApp has been bound to default_host.
[AUDIT ] CWWKT0016I: Web application available (default_host):
http://localhost:9080/ITSOTimerApp/
[AUDIT ] CWWKZ0001I: Application ITSOTimerApp started in 0.625 seconds.
[AUDIT ] CWWKF0012I: The server installed the following features:
[localConnector-1.0, jdbc-4.1, cdi-1.2, servlet-3.1, jndi-1.0, ejbLite-3.2].
[INFO ] CWWKF0008I: Feature update completed in 1.803 seconds.
[AUDIT ] CWWKF0011I: The server timerTest is ready to run a smarter planet.
Generating report...
```

Using persistent timers

Creating a persistent timer is similar to non-persistent, except for the server configuration, which is a bit more complex:

1. Start by creating a new project that holds your timer class.
2. Create a class with the code shown in Example 5-34. Make sure that you have a **persistent=true** or that you do not have a persistent element at all, as the default value for persistent element is true, if omitted. This timer runs on the tenth second of every minute.

Example 5-34 Persistent timer bean

```
@Singleton
@Startup
public class TimerBean {
    @PostConstruct
    public void init() {
        System.out.println("TimerBean initialized");
    }
    @Schedule(second="10", minute="*", hour="*")
    public void generateReport() {
        System.out.println("Generating report via persistent timer...");
    }
}
```

3. Configure the server to run the application with a non-persistent timer by using the following steps:
 - a. In the Servers view, expand your Liberty server and double-click **Server configuration**.
 - b. Add the **ejbPersistentTimer-3.2** feature, as shown in Figure 5-28.

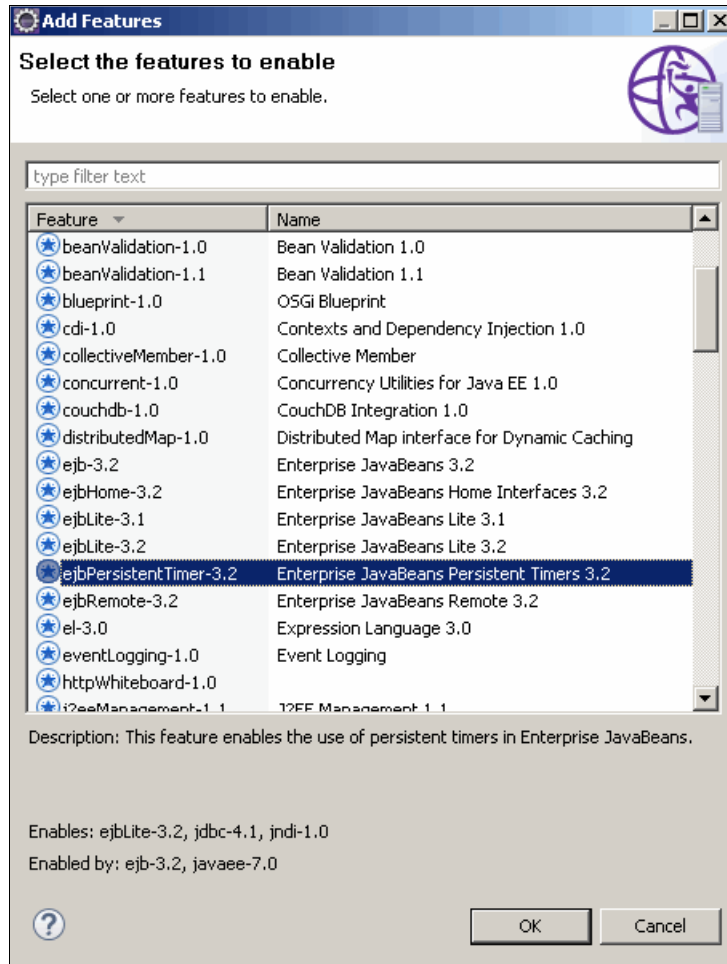


Figure 5-28 Adding the `ejbPersistentTimer` feature

You can also enable the `ejbPersistentTimer-3.2` feature manually, adding the snippet shown in Example 5-35 to `server.xml`.

Example 5-35 `ejbPersistentTimer` feature in `server.xml`

```
<featureManager>
  <feature>ejbPersistentTimer-3.2</feature>
</featureManager>
```

- c. Persistent timers require a database to store the timer details. In Liberty, persistent timers use the data source identified by the `DefaultDataSource` ID. Add default data source configuration to the `server.xml`, as shown in Example 5-36 on page 156. You can also add default data by using the graphical editor in a similar fashion (as shown in “Adding a data source using Liberty profile developer tools” on page 115).

Example 5-36 Default data source configuration using a Derby database

```
<dataSource id="DefaultDataSource">
  <jdbcDriver>
    <library name="derbyLibrary">
      <fileset dir="${shared.resource.dir}/derby"/>
    </library>
  </jdbcDriver>
```

```
<properties.derby.embedded createDatabase="create"
databaseName="EJBTimers"/>
</dataSource>
```

4. Deploy your application to the server. After a short time, you should see the output in the console, as shown in Example 5-37.

Example 5-37 Persistent timer running

```
[INFO ] CNTR4000I: The ITSOTimerApp.war EJB module in the ITSOTimerApp
application is starting.
[INFO ] CNTR0167I: The server is binding the com.ibm.itso.timers.TimerBean
interface of the TimerBean enterprise bean in the ITSOTimerApp.war module of
the ITSOTimerApp application. The binding location is:
java:global/ITSOTimerApp/TimerBean!com.ibm.itso.timers.TimerBean
[INFO ] DSRA8203I: Database product name : Apache Derby
[INFO ] DSRA8204I: Database product version : 10.8.2.3 - (1212722)
[INFO ] DSRA8205I: JDBC driver name : Apache Derby Embedded JDBC Driver
[INFO ] DSRA8206I: JDBC driver version : 10.8.2.3 - (1212722)
[INFO ] CNTR0219I: The server created 1 persistent automatic timer or timers
and 0 non-persistent automatic timer or timers for the ITSOTimerApp.war module.
TimerBean initialized
[INFO ] CNTR4001I: The ITSOTimerApp.war EJB module in the ITSOTimerApp
application has started successfully.
[INFO ] SRVE0169I: Loading Web Module: ITSOTimerApp.
[INFO ] SRVE0250I: Web Module ITSOTimerApp has been bound to default_host.
[AUDIT ] CWWKT0016I: Web application available (default_host):
http://localhost:9080/ITSOTimerApp/
[AUDIT ] CWWKZ0001I: Application ITSOTimerApp started in 2.859 seconds.
[AUDIT ] CWWKF0012I: The server installed the following features:
[ejbPersistentTimer-3.2, localConnector-1.0, jdbc-4.1, cdi-1.2, servlet-3.1,
jndi-1.0, ejbLite-3.2].
[INFO ] CWWKF0008I: Feature update completed in 4.141 seconds.
[AUDIT ] CWWKF0011I: The server timerTest is ready to run a Smarter Planet.
Generating report via persistent timer...
```

Using programmatic timers

Sometimes, timers need to be created on demand. This can be achieved by the `TimerService`. You create the timer using a `TimerService` API and a mark method to be invoked using the `@Timeout` annotation. Example 5-38 on page 157 shows sample code that creates a timer repeating every 10 seconds. The Returned `Timer` object can be used to query timer information (for example, when is the next execution of this timer) or cancel it (if you no longer want it to execute).

Refer to the `TimerService` Javadoc for more information; see the following website:

<http://docs.oracle.com/javaee/7/api/javax/ejb/TimerService.html>

Example 5-38 Programmatic timer example

```
@Stateless
public class ProgrammaticTimer {
    @Resource
    TimerService timerService;

    public Timer setupTimer() {
```

```

    System.out.println("Setting up timer");
    TimerConfig tConfig = new TimerConfig();
    tConfig.setPersistent(false);
    Timer timer = timerService.createIntervalTimer(new Date(), 10000, tConfig);
    return timer;
}
@Timeout
public void timeout() {
    System.out.println("Programmatic timer");
}
}

```

While creating the timer, you can, via TimerConfig object, attach any Serializable object to it that may provide some additional data to the timeout method. See Example 5-39 about how to pass and retrieve that additional information.

Example 5-39 Using additional data with timer

```

public Timer setupTimerWithInfo() {
    System.out.println("Setting up timer with Info");
    String timerInfo = "ImportantMonthlyReport";
    TimerConfig tConfig = new TimerConfig(timerInfo);
    tConfig.setPersistent(false);
    Timer timer = timerService.createIntervalTimer(new Date(), 10000, tConfig);
    return timer;
}
@Timeout
public void timeout(Timer timer) {
    // retrieve additional info
    System.out.println("Programmatic timer for: " + timer.getInfo());
}

```

5.3 Developing Java Message Service applications

By using the Java Message Service (JMS) API, you can send messages between Java EE applications and components. It allows you to develop applications with loosely coupled communication. JMS is supported by WebSphere Liberty Profile, but it is not supported by the WebSphere Liberty core edition.

The WebSphere Liberty profile provides an embedded messaging server that allows applications to act as a JMS server. Your applications can also act as JMS clients for another WebSphere Liberty profile instance, for an external WebSphere MQ server or for a third-party JMS provider when bundled with an application. For details about how to configure your application with Apache MQ, refer to chapter 7 of the IBM Redbooks publication *Configuring and Deploying Open Source with WebSphere Application Server Liberty Profile*, SG24-8194. WebSphere Liberty profile server supports both the point-to-point model and the publish and subscribe model.

Use the procedure that is outlined in the rest of this section to create a web application that uses the JMS to send and receive messages. If you are using Liberty core edition, you cannot compile this application in the Liberty profile developer tools because the API JAR files for JMS are not available.

In addition to the EJB lite 3.2 specification, Liberty profile server provides support for the MDB 3.2 specification. Message driven beans allow you to construct enterprise beans that operate asynchronously on messages that are received from other Java EE components.

5.3.1 What's new in JMS 2.0 API?

Java EE 7 specification updates JMS API from 1.1 to 2.0. The most important changes are noted in the following list:

- ▶ New simplified API. API includes JMSContext object, which can be used instead of Connection and Session objects.
- ▶ All objects that have to be closed implement Autoclosable interface.
- ▶ Resources can be configured and injected via annotations.

5.3.2 Developing applications using JMS 2.0 API

In this section, you create a simple web application that uses new JMS 2.0 API to send and receive messages. You also configure Liberty server to support messaging.

Creating an application

1. Start by creating a new dynamic web project that holds your application.
2. Create a servlet that will be used to send messages. The SenderServlet code is presented in Example 5-40. Observe injection of resources, the Queue and JMSContext, and how simple it is to send messages by using its interface.

Example 5-40 SenderServlet code

```
@WebServlet("/SenderServlet")
@Resource(name="jms/itsoQCF", lookup="jms/itsoQCF",
type=ConnectionFactory.class)
public class SenderServlet extends HttpServlet {
    @Inject
    @JMSConnectionFactory("java:comp/env/jms/itsoQCF")
    private JMSContext jmsContext;

    @Resource(lookup="jms/itsoQ")
    private Queue queue;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String message = "Test message " + new Date();
        if(request.getParameter("msg") != null) {
            message = request.getParameter("msg");
        }
        System.out.println("Sending message: " + message);
        jmsContext.createProducer().send(queue, message);
        System.out.println("Message Sent!");
        PrintWriter out = response.getWriter();
        out.println("Message Sent!");
    }
}
```

3. Create a second servlet to test the receiving of messages. This example uses synchronous receiving (see section 5.3.3, "Developing a message-driven bean (MDB)

application” on page 166, where you learn how to write a message-driven bean to read messages). Example 5-41 shows the ReceiverServlet. Again observe injection and usage of the new API to read messages.

Example 5-41 ReceiverServlet code

```
@WebServlet("/ReceiverServlet")
public class ReceiverServlet extends HttpServlet {

    @Inject
    @JMSConnectionFactory("jms/itsoQCF")
    private JMSContext jmsContext;

    @Resource(lookup="jms/itsoQ")
    private Queue queue;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        System.out.println("Receiving message...");
        PrintWriter out = response.getWriter();
        Message message = jmsContext.createConsumer(queue).receive(5000);
        if(message != null && message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            try {
                System.out.println("Received: " + textMessage.getText());
                out.println("Received: " + textMessage.getText());
            } catch (JMSException e) {
                out.println("Error: " + e.getMessage());
            }
        }
        else {
            System.out.println("No or unknown message");
            out.println("No or unknown message");
        }
    }
}
```

4. For injection of JMSContext to work, your application needs to be CDI enabled. You can do it by creating an empty beans.xml file in the WEB-INF folder in case of web modules or META-INF folder, in case of EJB or utility modules. Figure 5-29 on page 161 shows the project structure with servlets and beans.xml file.

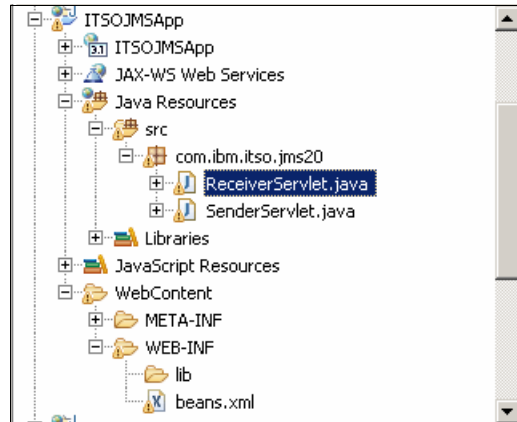


Figure 5-29 Project structure

Now the application code is ready. Next, prepare the server for it.

Configuring server for JMS application

This configuration uses an embedded JMS engine provided with Liberty. JMS support is not available in the Liberty Core edition; however, the developers version contains full support for JMS so that you can develop and test your JMS applications.

To configure Liberty server to support JMS, use the following steps:

1. In the **Servers** view, expand Liberty Server and double-click **Server Configuration**.
2. Start with adding features to the Feature Manager. You can do it either using a graphical editor or editing the server.xml directly. You need to add features that are shown in Example 5-42.

Example 5-42 Features required by JMS application

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>wasJmsServer-1.0</feature>
  <feature>wasJmsClient-2.0</feature>
  <feature>cdi-1.2</feature>
  <feature>jndi-1.0</feature>
</featureManager>
```

3. Next, configure the Messaging engine. Select **Server configuration** and click **Add**. Select **Messaging Engine**, as shown in Figure 5-30 on page 162, and click **OK**.

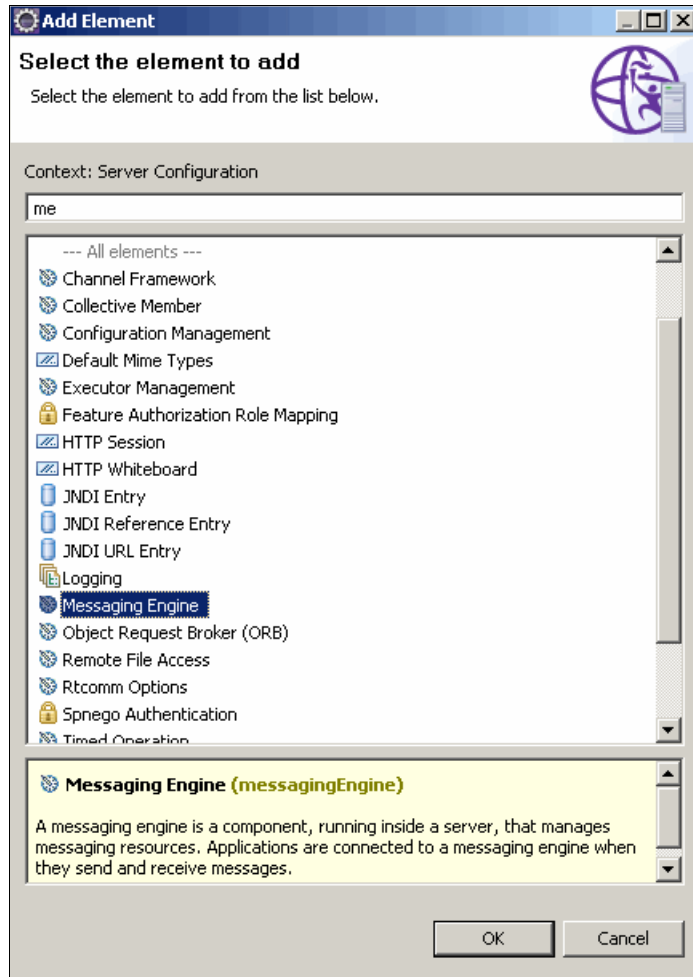


Figure 5-30 Adding the messaging engine

- Next, configure the queue. Select the recently added **Messaging Engine** element and click **Add**. Select **Queue**, as shown in Figure 5-31 on page 163, and click **OK**.

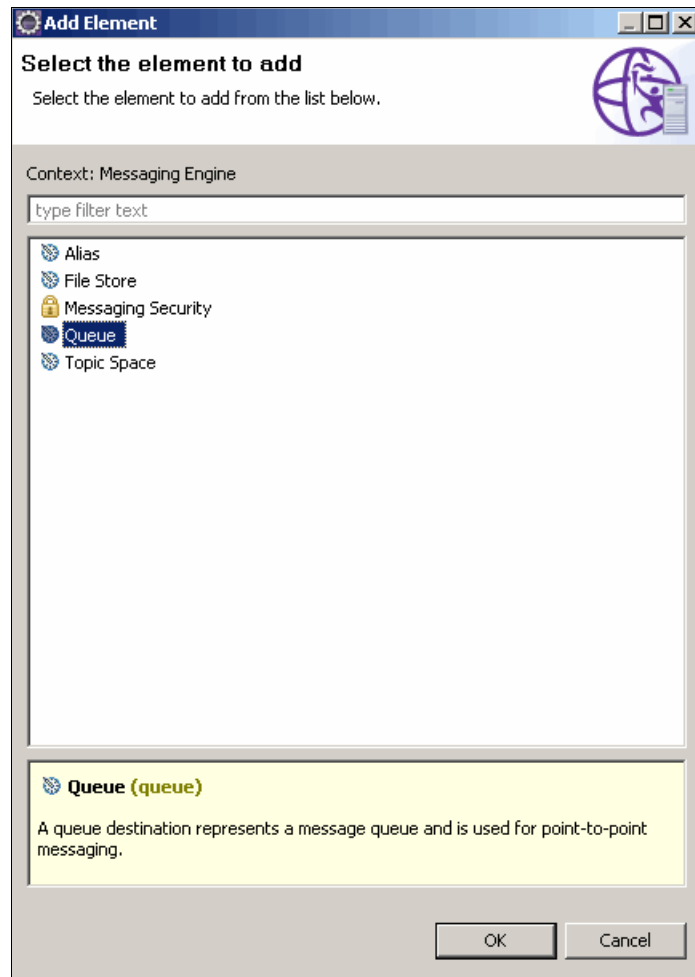


Figure 5-31 Adding queue

5. Select the just added Queue and specify the Queue name, in details, as itsoQ. See Figure 5-32.

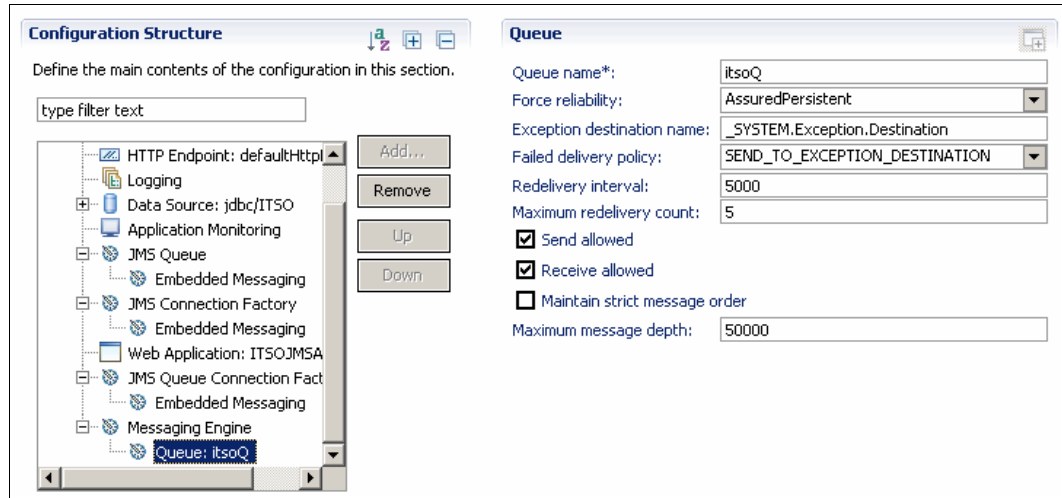


Figure 5-32 Queue details

The same configuration could be achieved by adding text from Example 5-43 to the server.xml file.

Example 5-43 Messaging engine configuration in server.xml

```
<messagingEngine>
  <queue id="itsoQ"></queue>
</messagingEngine>
```

6. Configure required JMS resources (the connection factory and queue). Start with the connection factory.

7. Select **Server configuration**; click **Add**. Start typing JMS and select **JMS Connection Factory**, as shown in Figure 5-33, and click **OK**.

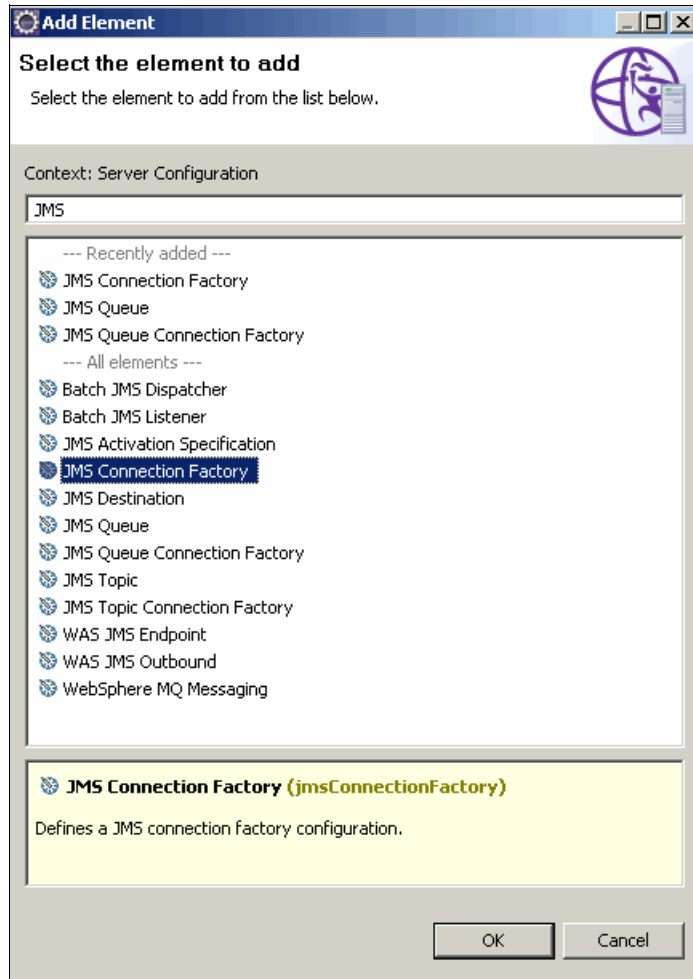


Figure 5-33 Adding connection factory

8. Select the recently added **JMS Connection Factory**, enter `jms/itsoQCF` as the JNDI name, and click **Add**. Select **Embedded Messaging** and click **OK**.
9. In a similar way, add **JMS Queue**. Enter `jms/itsoQ` as JNDI name, and `itsoQ` as Queue name in the Embedded Messaging configuration, as shown in Figure 5-34 on page 165.

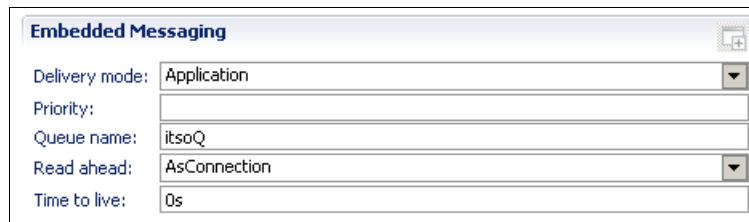


Figure 5-34 Queue configuration details

10. Similar configuration of JMS resources could be done by adding Example 5-44 fragment to the `server.xml` file.

Example 5-44 JMS resources configuration in server.xml

```
<jmsConnectionFactory jndiName="jms/itsoQCF">
  <properties.wasJms/>
</jmsConnectionFactory>
<jmsQueue jndiName="jms/itsoQ">
  <properties.wasJms queueName="itsoQ"/>
</jmsQueue>
```

11. Your configuration is done. Save **Server Configuration** and close the editor.

Testing JMS application

To test the JMS application, follow these steps:

1. Start the server and deploy the sample application to it.
2. First invoke SenderServlet, either from Workspace or by using the following URL:
`http://localhost:9080/ITSOJMSApp/SenderServlet`.
You should see **Message sent!** text in the browser.
3. Now invoke ReceiverServlet, either from Workspace or by using the following URL:
`http://localhost:9080/ITSOJMSApp/ReceiverServlet`. You should see **Received: Test message Wed May 13 01:59:43 CEST 2015** text in the browser.

5.3.3 Developing a message-driven bean (MDB) application

The Liberty profile developer tools simplify the development of message-driven beans by providing wizards for creating message-driven beans and for creating related JMS resources. You extend the application from the previous section with MDB that will be invoked by the container when a new message arrives in the queue.

Creating the message-driven bean

To create a message driven bean, complete the following steps:

1. Select the ITSOJMSApp project. By using the menu, click **File** → **New** → **Message Driven Bean**.

2. In the message-driven bean wizard, enter the package name `com.ibm.itso.jms20` and the class name as `SimpleMDB`. Click **Finish**.
3. To finish implementing the `ExamplePubSubMDB` class, modify the `onMessage` method so that it prints the message that is received by the console. The completed message-driven bean class is shown in Example 5-45.

Example 5-45 Complete `SimpleMDB` code

```

@MessageDriven(name="SimpleMDB")
public class SimpleMDB implements MessageListener {
    public void onMessage(Message message) {
        try {
            System.out.println("MDB: " + ((TextMessage)message).getText());
        } catch (JMSEException e) {
            e.printStackTrace();
        }
    }
}

```

Creating message-driven bean resources

This example uses the resources that are created in the JMS example in “Configuring server for JMS application” on page 161, including the messaging engine, queue, and JMS Queue. For this message-driven bean example, you must also create a JMS activation specification and properties.

To create a JMS activation specification, complete the following steps:

1. In the Servers view, expand the **Liberty Server** and double-click the **Server Configuration**.
2. Select **Server Configuration** and click **Add**. Start typing **JMS** and select **JMS Activation Specification**.
3. In the details window, specify a value for the ID field. The value must be in the format *applicationName/moduleName/beanName*, where *applicationName* is the name of the EAR application. Only *applicationName* is needed, if the application is packed as EAR. *ApplicationName* should be omitted if the application is a WAR or stand-alone EJB JAR file. The *moduleName* is the name of the module in which the message-driven bean is packaged (without extension). The *beanName* is the ejb-name of the enterprise bean. For this example, you use the application packed as WAR, so enter the value `ITSOJMSApp/SimpleMDB`.

You also must add properties to the JMS activation specification to indicate that you are using the embedded messaging engine and to specify the JMS Queue to be used. To do this, complete the following steps:

1. Click **JMS Activation Specification** in the server configuration and click **Add**.
2. Select **WebSphere Embedded JMS Activation Specification** in the dialog box.
3. On the details window for the properties, enter `jmsItsoQ` in the Destination reference field. Set ID in the JMS Queue (changing from `itsoQ` to `jmsItsoQ`), if the ID was not defined earlier. ID in the queue configuration is required.

Enabling features

To use the message-driven beans, you need to enable the `mdb-3.2` feature. To add the `mdb-3.2` feature, complete the following steps:

1. Select **Feature Manager** in the server configuration.

2. Click **Add**, and select the **mdb-3.2 feature**.
3. It is assumed that other features required by ITSOJMSApp are already defined. If they are not, you need the features that are shown in Example 5-46.

Example 5-46 Features required to run MDB application

```
<featureManager>
  <feature>wasJmsServer-1.0</feature>
  <feature>wasJmsClient-2.0</feature>
  <feature>cdi-1.2</feature>
  <feature>mdb-3.2</feature>
</featureManager>
```

Testing the message-driven bean

To test the message-driven bean, use the SenderServlet servlet from 5.3.2, “Developing applications using JMS 2.0 API” on page 159. Point your browser to the following link to invoke servlet that will send a message, which will be received by the message-driven bean:

<http://localhost:9080/ITSOJMSApp/SenderServlet>

In the Console view, you should see the output shown in Example 5-47.

Example 5-47 Output from running MDB test

```
[INFO    ] SRVE0242I: [ITSOJMSApp] [/ITSOJMSApp]
[com.ibm.itso.jms20.SenderServlet]: Initialization successful.
Sending message:  Test message Mon May 18 00:51:46 CEST 2015
Message Sent!
MDB: Test message Mon May 18 00:51:46 CEST 2015
```

5.4 Developing applications using asynchronous and concurrent features

Java EE 7 specification allows the application to invoke some of its logic asynchronously, as background tasks, and continue its main processing. Later, when the result is ready, it can be retrieved and returned to the client application.

Asynchronous call is usually used for long-running tasks, or tasks that require waiting for a resource that is needed to generate the response. Such operations limit the scalability of the application. Asynchronous call is running in a separate thread, which allows a return of the thread associated with the original request immediately to the container and then be available for processing requests from the next client.

Asynchronous processing can be achieved using the following methods:

- ▶ Asynchronous Servlets and Filters
- ▶ Asynchronous methods in Session beans
- ▶ Usage of Concurrent API

Each of these methods is briefly discussed in the next sections.

5.4.1 Developing using @Asynchronous in servlets and filters

Java EE provides asynchronous processing support for servlets and filters. You create a simple servlet that prepares a response in asynchronous call.

1. Start by creating a new dynamic web project.
2. In the project, start the New Servlet wizard, and enter `AsyncServlet` as the class name and click **Next**.
3. **Check** Asynchronous Support, as shown in Figure 5-35, and click **Finish**.

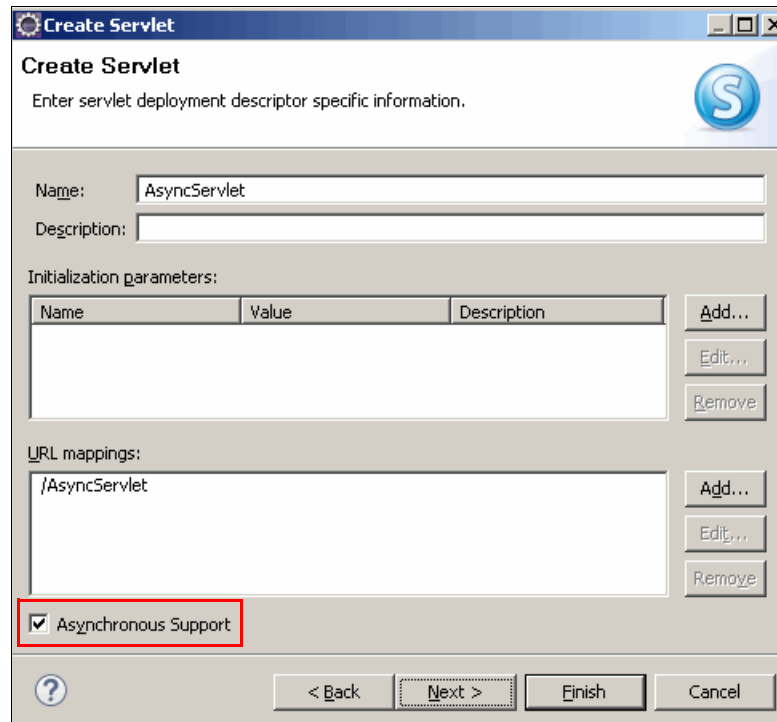


Figure 5-35 Enabling Asynchronous support in the servlet

4. Notice that the created servlet has `asyncSupported=true`, as shown in Example 5-48.

Example 5-48 `AsyncServlet` definition

```
@WebServlet(asyncSupported = true, urlPatterns = { "/AsyncServlet" })  
public class AsyncServlet extends HttpServlet {
```

Notice: If the request to the asynchronous servlet goes through servlet filters, they also need to have asynchronous support enabled.

5. Start to implement the `doGet` method. To put the request in the asynchronous mode and initialize `AsyncContext`, you call the `request.startAsync()` method.
6. Asynchronous tasks are represented as objects implementing the *Runnable* interface. You can either create a separate class for the task or instantiate the task as anonymous class. You need to pass the runnable task as parameter to the `asyncContext.start()` method. In this simple example, you use anonymous class as shown in Example 5-49 on page 170. If you want to use a separate class, remember to pass the `asyncContext` object to it. The `run()` method of the class just prints text to the servlet response and to the system out log. To simulate a long call, there are some calls to the `Thread.sleep(3000)` in the method.

Example 5-49 Implementing asynchronous method

```
    asyncContext.start(new Runnable() {
        public void run() {
            try {
                String jobName = asyncContext.getRequest().getParameter("name");
                if(jobName == null) jobName = "TestJob";
                ServletResponse response = asyncContext.getResponse();
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                out.println("<H2>Async servlet doing job: " + jobName + "</H2>");
                out.println("Doing async job...<BR>");
                out.flush();
                System.out.println("Doing async job...");
                Thread.sleep(3000);
                out.println("Doing async job...<BR>");
                out.flush();
                System.out.println("Doing async job...");
                Thread.sleep(3000);
                out.println("Sending response...<BR>");
                out.flush();
                System.out.println("Sending response");
                asyncContext.complete();
            }
            catch(Exception e) {
                e.printStackTrace();
            }
        }
    });
```

7. Call the `asyncContext.complete()` method to notify the container that the response should be returned. The full servlet code is shown in Example 5-50.

Example 5-50 AsyncServlet code

```
@WebServlet(asyncSupported = true, urlPatterns = { "/AsyncServlet" })
public class AsyncServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        System.out.println("Before async");
        final AsyncContext asyncContext = request.startAsync();
        asyncContext.start(new Runnable() {
            public void run() {
                try {
                    String jobName = asyncContext.getRequest().getParameter("name");
                    if(jobName == null) jobName = "TestJob";
                    ServletResponse response = asyncContext.getResponse();
                    response.setContentType("text/html");
                    PrintWriter out = response.getWriter();
                    out.println("<H2>Async servlet doing job: " + jobName + "</H2>");
                    out.println("Doing async job...<BR>");
                    out.flush();
                    System.out.println("Doing async job...");
                    Thread.sleep(3000);
                    out.println("Doing async job...<BR>");
                    out.flush();
                    System.out.println("Doing async job...");
                }
            }
        });
    }
}
```

```

        Thread.sleep(3000);
        out.println("Sending response...<BR>");
        out.flush();
        System.out.println("Sending response");
        asyncContext.complete();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
});
try {
    // sleep added to show 2 different threads executing
    Thread.sleep(3000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
System.out.println("Ending main thread");
}
}
}

```

8. When you run this example in the message.log file, you can observe that it is executed by using two different threads (as shown in Example 5-51).

Example 5-51 Output in the message.log

```

[5/15/15 14:26:27:989 CEST] 0000005d SystemOut 0 Before async
[5/15/15 14:26:27:989 CEST] 0000005b SystemOut 0 Doing async job...
[5/15/15 14:26:30:989 CEST] 0000005b SystemOut 0 Doing async job...
[5/15/15 14:26:30:989 CEST] 0000005d SystemOut 0 Ending main thread
[5/15/15 14:26:33:989 CEST] 0000005b SystemOut 0 Sending response

```

5.4.2 Developing using @Asynchronous in session beans

Until Java EE 7, all calls to session beans were synchronous, and control was not returned to the client until the method has ended. With the introduction of asynchronous methods in session beans, control is returned immediately and the client can execute other logic while the bean is running. Asynchronous method returns an implementation of the `java.util.concurrent.Future<V>` interface, which allows the client to check: If the method has completed, retrieved the result, or requested computation to be canceled.

While defining asynchronous beans, specification allows to either mark a single method or class with `@Asynchronous` annotation. If the class is marked, all business methods are asynchronous. For simplicity, use annotation defined on the class level.

To create a session bean with asynchronous support, use the following steps:

1. Create a new Session bean by selecting **File** → **New** → **Session bean**. Enter AsyncBean as a class name and **check** Asynchronous, as shown in Figure 5-36 on page 172.

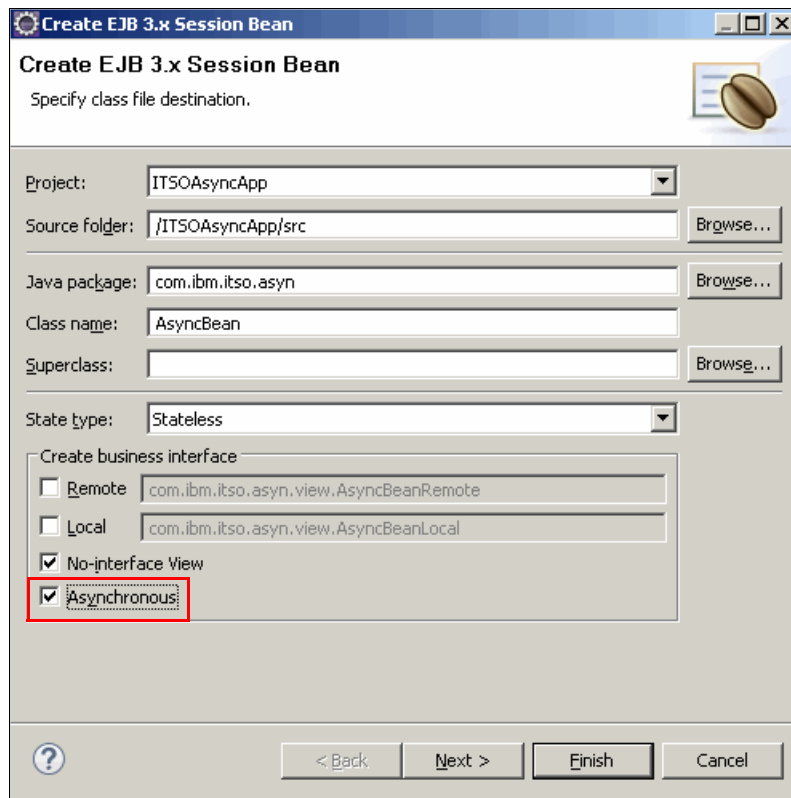


Figure 5-36 Creating asynchronous session bean

2. Observe @Asynchronous annotation in the generated class in Example 5-52.

Example 5-52 Asynchronous session bean

```
@Stateless
@LocalBean
@Asynchronous
public class AsyncBean {
```

3. You implement a simple method that does some computation and returns AsyncResult<String> object, which implements Future interface, as shown in Example 5-53.

Example 5-53 Sample asynchronous method

```
public Future<String> longRunningCall() {
    try {
        System.out.println("In logng running method...");
        Thread.sleep(10000);
        System.out.println("In logng running method...Ending");
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return new AsyncResult<String>("Done " + new Date());
}
```

```
}
```

4. As a test client, a servlet is created. Servlet invokes a session bean method and stores the returned Future object in session. Later, servlet can check, by using the `Future.isDone()` method, if the result is ready, and retrieve the result using the `Future.get()` method. The complete `doGet()` method of the servlet is shown in Example 5-54.

Example 5-54 Servlet `doGet` method

```
@EJB
AsyncBean bean;

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    HttpSession httpSession = request.getSession();

    if(request.getParameter("create") != null) {
        out.println("Calling bean");
        Future<String> result = bean.longRunningCall();
        out.println("isDone? " + result.isDone());
        httpSession.setAttribute("result", result);
    }
    else {
        Future<String> result =
            (Future<String>) httpSession.getAttribute("result");
        if(result != null) {
            out.println("isDone? " + result.isDone());
            if(result.isDone()) {
                try {
                    out.println("Result: " + result.get());
                    httpSession.removeAttribute("result");
                } catch (InterruptedException | ExecutionException e) {
                    e.printStackTrace();
                }
            }
            else {
                out.println("Not ready yet");
            }
        }
        else {
            out.println("Not started yet");
        }
    }
}
}
```

5.4.3 Developing using Concurrent API

Sometimes applications need to do some long running computation or schedule a task to be at a specific time on demand. This usually is done by creating new threads or using third-party libraries. It is a Java EE best practice to not create unmanaged threads. Java EE 7 provides a standardized way to obtain managed threads that can run with context provided by the container using concurrent API.

Concurrent API provided by the server offers the following classes to support asynchronous processing:

- ▶ **Managed executors:** Provides a way to start asynchronous tasks as `Runnable` or `Callable` objects within an application server environment with propagated context.

You see an example of managed executor usage later in this chapter.

For more information about configuring and using managed executors, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_config_managedexecutor.html?lang=en-us

- ▶ **Managed scheduled executors:** Can schedule a task to run at a specific time or interval. For more information about configuring and using scheduled executors, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_config_scheduledexecutor.html?lang=en-us

- ▶ **Managed thread factory:** Allows to obtain container-managed thread for execution of custom code. Code runs within the context propagated from the thread that looked up or injected the managed thread factory.

For more information about configuring and using managed thread factory, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_config_threadfactory.html?lang=en

- ▶ **Thread context service:** Allows to capture a managed thread context and apply it to invocations of specified interface methods on any thread. For more information about configuring and using context service, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_config_contextservice.html?lang=en

Using Managed Executor

In this section, you learn how to use managed executor and how to configure a server to support that. You access executor via `ExecutorService`. To create a class that uses executor service, use the following steps:

1. In the project, start the New Servlet wizard, and enter `ConcurrentServlet` as the class name and click **Finish**.
2. Inject into the servlet, the **`ExecutorService`**, as shown in Example 5-55. In the `@Resource` annotation, specify the JNDI name of the factory. It is configured later in the server.

Example 5-55 Injecting ExecutorService

```
@WebServlet("/ConcurrentServlet")
public class ConcurrentServlet extends HttpServlet {
    @Resource(lookup="concurrent/executorService")
    ExecutorService executorService;
}
```

3. In the `doGet` method, use the `executorService` to submit a `Runnable` object to it. This is shown in Example 5-56.

Example 5-56 Submitting a task for executor

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
    System.out.println("In doGet - creating task");

    executorService.submit(new Runnable() {
        @Override
        public void run() {
            System.out.println("in task");
            // your Runnable code
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Finishing task");
        }
    });
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Returning servlet method");
    PrintWriter out = response.getWriter();
    out.println("Servlet finished, task still working");
}
```

4. This simple servlet shows that you can finish your web request, while the newly created managed task is still executing. The sample output (the contents of the `message.log` file) is shown in Example 5-57.

Example 5-57 Output of running servlet

```
[5/17/15 2:09:15:529 CEST] 00000160 com.ibm.ws.webcontainer.servlet I
SRVE0242I: [ITSOAsyncApp] [/ITSOAsyncApp]
[com.ibm.itso.async.ConcurrentServlet]: Initialization successful.
[5/17/15 2:09:15:529 CEST] 00000160 SystemOut 0 In get - creating task
[5/17/15 2:09:15:529 CEST] 0000016a SystemOut 0 in task
[5/17/15 2:09:17:529 CEST] 00000160 SystemOut 0 Returning servlet method
[5/17/15 2:09:20:545 CEST] 0000016a SystemOut 0 Finishing task
```

Configuring the server for using concurrent API

To be able to use resources like thread factories or executors, you have to configure them in `server.xml` and enable the `concurrent-1.0` feature. Use the following steps to configure the server:

1. In the Servers view, expand **Liberty server** and double-click **Server Configuration**.

2. Select **Server Configuration**, click **Add**, and start typing **Managed**. Select **Managed Executor**, as shown in Figure 5-37, and click **OK**. As you can see, other objects that were mentioned in this chapter are available for configuration.

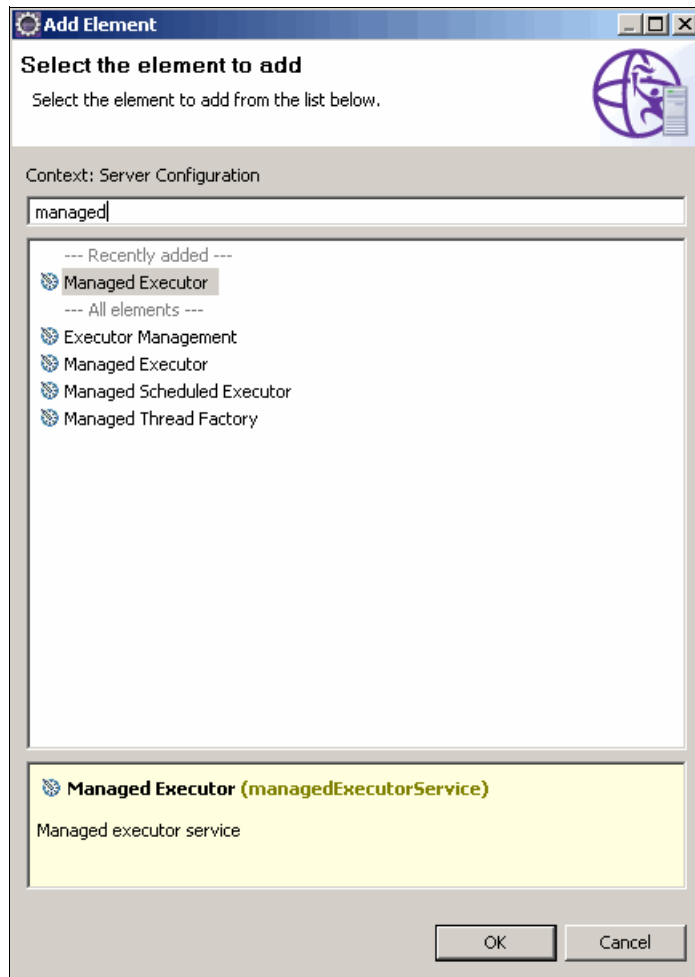


Figure 5-37 Adding Managed Executor to the configuration

3. Accept the prompt to enable the concurrent-1.0 feature, as shown in Figure 5-38 on page 177.

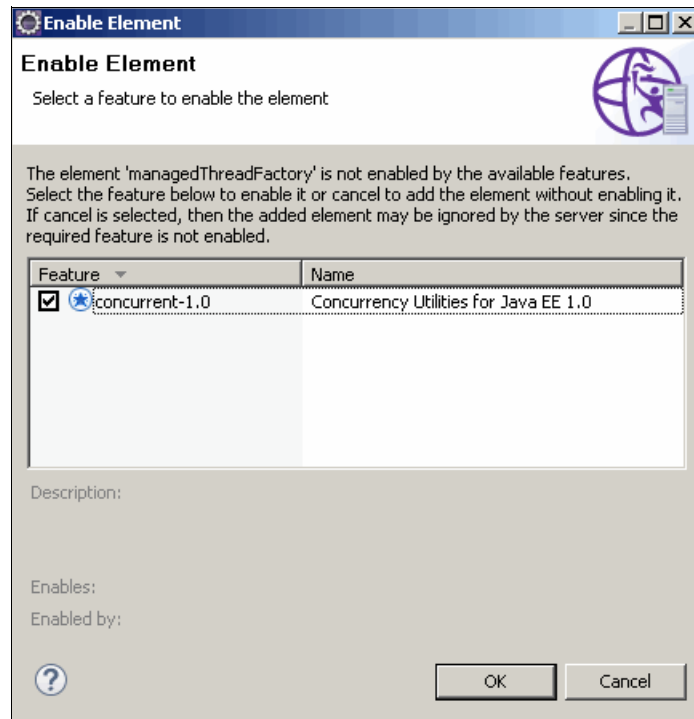


Figure 5-38 Enabling concurrent-1.0 feature

4. Finish configuring the executor by entering `concurrent/executorService` in the JNDI name.
5. Save the changes. The server is ready to run the example. After deployment, you can invoke it by using `http://localhost:9080/ITS0AsyncApp/ConcurrentServlet`.

5.5 Developing applications using JavaMail

Liberty profile server now supports JavaMail features as required by Java EE 7 specification. You can send and receive emails in applications running on a Liberty server. MailSession object can be injected in application by using annotation or by JNDI lookup.

5.5.1 Writing, testing, and deploying the JavaMail sample application

Use the following steps to write a simple JavaMail application that can send a mail to a valid recipient through the server:

1. Create a new web application.
2. Create a new servlet with the content shown in Example 5-58. Make sure to use the correct “toAddress” value.

Example 5-58 Mail sending servlet

```
@WebServlet("/MailSender")
public class MailSender extends HttpServlet {
    @Resource(lookup="mail/itsMailSession")
```

```

Session mailSession;

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    System.out.println("Sending test email");
    String toAddress = "valid@email.address";
    String message = "Test message";
    if(request.getParameter("msg") != null) {
        message = request.getParameter("msg");
    }
    Message mail = new MimeMessage(mailSession);
    try {
        mail.setRecipient(Message.RecipientType.TO, new
InternetAddress(toAddress));
        mail.setSubject("Test subject");
        mail.setSentDate(new Date());
        mail.setText(message);
        Transport.send(mail);
        System.out.println("Message sent successfully");
    } catch (MessagingException e) {
        e.printStackTrace();
    }
}
}
}

```

3. Configure a mail session in the Liberty server:
 - a. In the Servers view, expand **Liberty server** and double-click **Server Configuration**.
 - b. In the Configuration Structure section, highlight **Server Configuration** and click **Add**.
 - c. In the Select the element to add section, start typing mail and select **Mail Session Object**, as shown in Figure 5-39 on page 179.

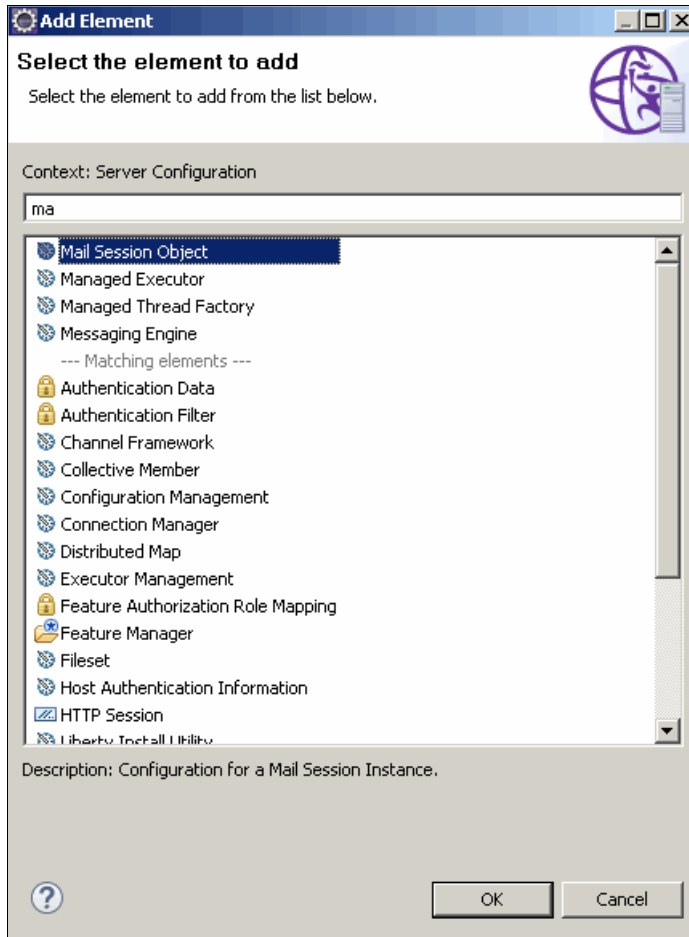


Figure 5-39 Adding a Mail Session element

- d. Provide Mail Session parameters: Mail session ID, JNDI name, description, mail server host, valid user name, and password for your mail server. You can also provide a default “From” address. The parameters are shown in Figure 5-40.

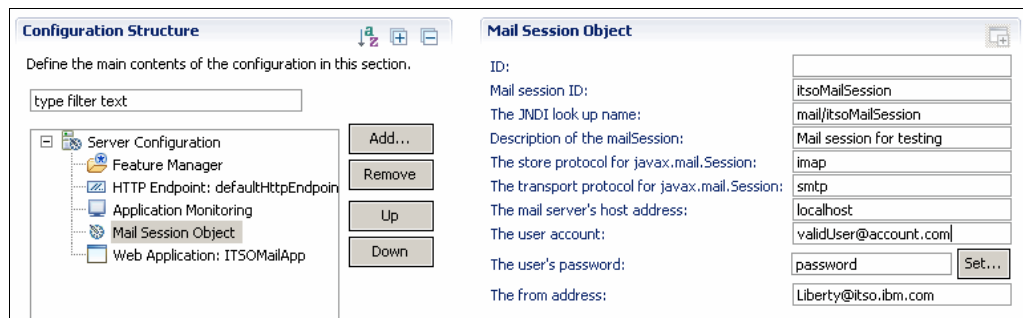


Figure 5-40 Mail Session details

Instead of using tools, the same outcome can be achieved by editing the `server.xml` file directly. Example 5-59 shows part of the `server.xml` file that is related to the configuration previously done.

Example 5-59 Mail Session in server.xml

```
<mailSession
  description="Mail session for testing"
  from="Liberty2@itso.ibm.com"
  host="localhost"
  jndiName="mail/itsoMailSession"
  mailSessionID="itsoMailSession"
  user="validUser@account.com"
  password="password"/>
```

4. Add `servlet-3.1`, `cdi-1.2`, and `javaMail-1.5` features to the Liberty `server.xml` file as shown in Figure 5-41.

```
<featureManager>
  <feature>servlet-3.1</feature>
  <feature>javaMail-1.5</feature>
  <feature>cdi-1.2</feature>
</featureManager>
```

Figure 5-41 Enabling features in Liberty server.xml

5. Deploy the application on the Liberty profile server by copying it to the `dropins` folder or running directly from `Workspace`.
6. Use the following URL to invoke the `MailServlet` on the Liberty profile server:
`http://<hostname>:<httpport>/ITSOMailApp/MailSender`

A message should now be sent to the recipient mentioned in the servlet using the James mail server that is running outside of Liberty profile server.



Configuring application security

Security is an essential component of any enterprise-level application. In this chapter, we provide you with a basic introduction to security using the Liberty profile.

The chapter contains the following sections:

- ▶ Enabling SSL
- ▶ HTTPS redirect
- ▶ Form login
- ▶ Securing JMS applications
- ▶ JAX-WS security
- ▶ Securing NoSQL applications

It is beyond the scope of this book to describe all aspects of security and how to configure them. For more information about security and how to implement specific security models not covered by this chapter (such as LTPA or SSO), see the IBM Knowledge Center at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_sec_webappsecurity.html?cp=SSAW57_8.5.5%2F1-3-11-0-4-6

6.1 Enabling SSL

You can configure the Liberty profile server to provide secure communication between a client and the server by enabling SSL. To establish secure communication, a certificate and an SSL configuration must be specified.

The keystore and certificate can be created either by using the WebSphere developer tools or from the command line. For development, you usually use a self-signed certificate, as shown in our examples later in this chapter.

Important: On publicly facing production servers, import a certificate from a trusted provider to the server keystore rather than using a self-signed certificate.

6.1.1 Configuration using the WebSphere developer tools

Use the WebSphere developer tools to add the keystore and self-signed certificate to your server using the following steps:

1. Open the Servers view in the workbench.
2. Right-click your **server** and select **Utilities** → **Create SSL Certificate**, as shown in Figure 6-1.

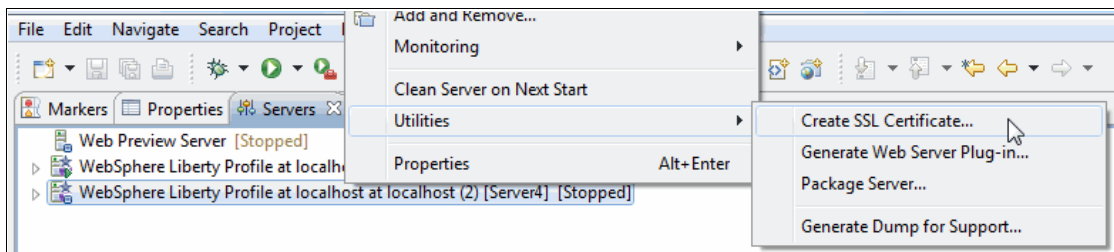


Figure 6-1 Selecting to create the SSL Certificate using WebSphere developer tools

- You are prompted to create a password for your keystore and optionally set the validity period or subject for the certificate, as shown in Figure 6-2 on page 183.

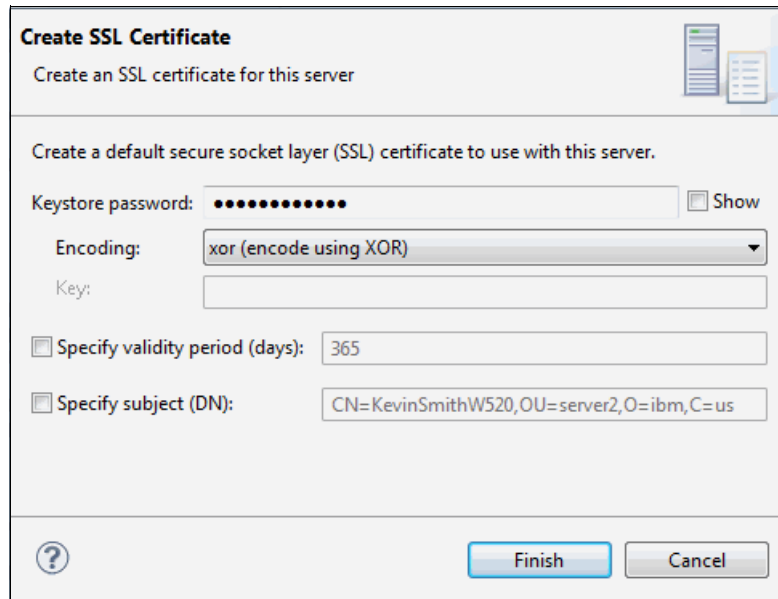


Figure 6-2 Entering values for the keystore and certificate

- Enter the required values and click **Finish**. The Console window shows the output of the command, as shown in Figure 6-3.

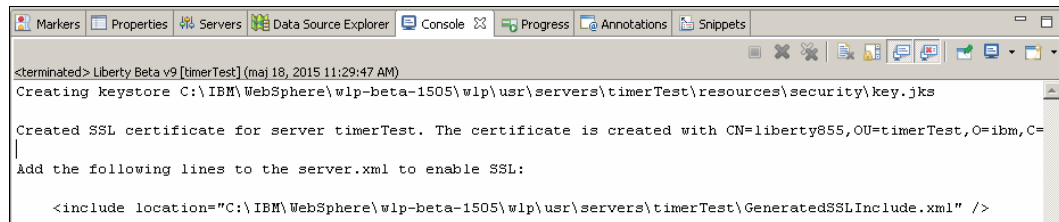


Figure 6-3 Output of the Create SSL Certificate command

- The output shows the line that should be added into the server configuration (server.xml). The line specifies to include GeneratedSSLInclude.xml file with SSL configuration (as shown in Example 6-1). The GeneratedSSLInclude.xml file contains configuration of the default keystore and enablement of the ssl-1.0 feature.

Example 6-1 GeneratedSSLInclude.xml file contents.

```
<?xml version="1.0" encoding="UTF-8" ?>
<server description="This file was generated by the 'securityUtility
createSSLCertificate' command on 2015-05-18 11:29:48 CEST.">
  <featureManager>
    <feature>ssl-1.0</feature>
  </featureManager>
  <keyStore id="defaultKeyStore" password="{xor}Lz4sLCgwLTs=" />
</server>
```

6. If the line with the `include` command was not added to the `server.xml` automatically, you can paste the required line directly into the `server.xml` configuration file (in which case your configuration is complete and you can proceed to “HTTPS redirect”). Alternatively, use the Design view to be guided through the configuration, as shown in the following steps:
 - a. In the Servers view, expand the **server** and double-click **Server configuration**.
 - b. In the opened editor, select **Server Configuration** and click **Add**. Start typing `Include` and select the **Include** element, as shown in Figure 6-4. Then, click **OK**.

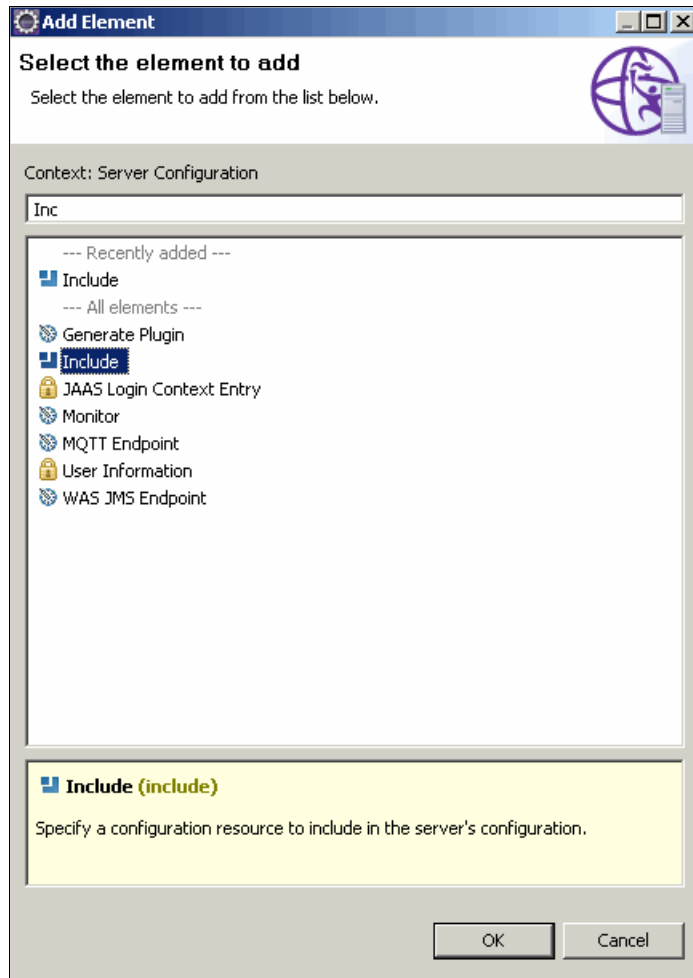


Figure 6-4 Adding Include element

- c. Select added **Include** element, and beside the `Location` field, click **Browse**. Next, select the `GeneratedSSLInclude.xml` file and click **OK**.
 - d. Your configuration is now complete. Save the changes to the server configuration. The server, if started, detects the new settings and the SSL port is opened.
7. If you have applications that are installed on your server, you can now access them by specifying `https://` and pointing your browser to the `https` port defined in your server configuration, `httpEndpoint`.

6.1.2 Configuration using the command line

To create the keystore and generate the self-signed certificate using the command line, you must use the `securityUtility` command that is in `${wlp.install.dir}/bin`. Example 6-2 shows the command and the expected output.

Example 6-2 Using the securityUtility command

```
C:\Liberty\v85Runtime\bin>securityUtility createSSLCertificate
--server=myThirdServer --password=passw0rd --validity=365
--subject="CN=liberty,O=IBM,C=US"
```

```
Creating keystore
```

```
C:\Liberty\v85Runtime\usr\servers\myThirdServer\resources\security\key.jks
```

```
Created SSL certificate for server myThirdServer
```

Add the following lines to the `server.xml` to enable SSL:

```
<featureManager>
  <feature>ssl-1.0</feature>
</featureManager>
<keyStore id="defaultKeyStore" password="{xor}Lz4sLChvLTs=" />
```

The output from the command contains the information to be added into your server configuration to enable SSL. Copy and paste this output into your `server.xml` file. The server automatically detects the changes and activates the SSL port.

6.2 HTTPS redirect

With SSL enabled, the client can, in most cases, still access the application on both the secure and nonsecure ports. HTTPS redirect allows you to always direct the client to the secure HTTPS port even if the client requested the standard nonsecure port.

This section assumes that you are starting with the web application created in Chapter 3, “Developing and deploying web applications” on page 59. If you do not have this application in your project workspace, you can find a completed version included in the download material for this book. For more information see Appendix A, “Additional material” on page 257. As well, you can use any other web application that you currently have in your workspace.

To access and edit the available web applications to work with HTTPS redirect, complete the following steps:

1. Security settings for web applications are defined in the `web.xml` file. However, this file is not required, so it is possible that you do not have one in your project. Select your project, right-click, and choose **Java EE Tools Generate** → **Deployment Descriptor Stub**.
2. In the Project Explorer, expand your web application and double-click the **deployment descriptor**.
3. To enable HTTPS redirect, add a security constraint to your application. To do this, select **Web Application** in the Overview section and then click **Add**.
4. In the Add Item window, start typing Security and select **Security Constraint**, then click **OK**.

- The Security Constraint now appears in the Overview section. Next, define the Web Resource Collection for the Security Constraint. This identifies which addresses are covered by the Security Constraint. Expand the Security Constraint in the Overview section and select **Web Resource Collection**.
- The details window, which is located to the right of the Editor, allows you to enter the details of the Web Resources. Give the Web Resource Collection a name and add in the URL Patterns to be used, as shown in Figure 6-5. Use pattern /* to protect all resources in the application.

The screenshot shows a 'Details' window with the following fields and controls:

- Web Resource Name*:** Text input field containing 'allresources'.
- URL Pattern*:** Text input field containing '/*'.
- Buttons:** 'Add', 'Remove', 'Up', and 'Down' buttons are positioned to the right of the URL Pattern field.
- Description:** A text area at the bottom of the window.

Figure 6-5 Entering details for the Web Resource collection

- Next, specify the details of the Security Constraint. In the Overview section, select **Security Constraint**. The details window now allows you to define the Security Constraint. Give the Security Constraint a name. Under the User Data Constraint section, select **CONFIDENTIAL** for the transport guarantee, as shown in Figure 6-6. This allows only secure communication with the web resource collection.

The screenshot shows a 'Details' window for a Security Constraint with the following sections and fields:

- Display Name:** Text input field containing 'Security constraint'.
- Authorization Constraint (optional):** A collapsed section that is currently expanded, showing:
 - Role Name:** Text input field.
 - Buttons:** 'Add', 'Remove', 'Up', and 'Down' buttons to the right of the Role Name field.
 - Description:** Text area below the Role Name field.
- User Data Constraint (optional):** A collapsed section that is currently expanded, showing:
 - Transport Guarantee*:** Dropdown menu set to 'CONFIDENTIAL'.
 - Description:** Text area below the Transport Guarantee field.

Figure 6-6 Entering details for Security constraint

8. This example currently applies the HTTPS redirect to all communications with the addresses defined in the Web Resource Collection. To apply this redirect to only specific HTTP methods, right-click the **Web Resource Collection** in the Overview section and select **Add** → **HTTP Method**. You restrict HTTPS to only **GET** requests by entering just **GET** for the HTTP Method. However, it is recommended that you use the same settings for all HTTP methods.
9. The next step is to enable application security. In the server view, expand the server until you see the Feature Manager entry. Double-click the **Feature Manager** or right-click and select **Open**.
10. The Feature Manager window opens. Click **Add**, select **appSecurity-2.0** from the list, and click **OK**.
11. HTTPS redirect is now fully configured. Save your changes and the WebSphere Developer Tool automatically updates the server with the new version of your application. If you now attempt to access the application at `http://localhost:9080/yourContext`, you notice that you are always redirected to `https://localhost:9443/yourContext`.

6.3 Form login

One of the most common ways to provide an authentication method to users of a web application is to use *form-based login*. This presents the users with a form where they can provide their login credentials. This section describes how to update your application and server to provide the form-based login.

This section expands on the HTTPS redirect example in 6.2, “HTTPS redirect” on page 185.

6.3.1 Defining the basic user registry

The first step in configuring form login is to create and define the actual registry that is used for authentication. In our example, we use a Basic User Registry. However, the Liberty profile does support other registries, including LDAP and custom registry. The following website provides more information about configuring the Liberty profile to work with these other registries:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_sec_authenticating.html?cp=SSAW57_8.5.5%2F1-3-11-0-4-2

Use the following steps to create and define the registry for authentication using a Basic User Registry:

1. Open the server view in your workbench and double-click the **Server Configuration** for your server. This opens your `server.xml` file. Switch to the design tab, if it is not already selected.
2. In the Configuration Structure section located to the left, select the **Feature Manager**. The Feature Manager details are displayed to the right.
3. If the `appSecurity-2.0` feature is not already in the list of features, click **Add** and select the **appSecurity-2.0** feature.
4. Add the basic registry that is used for authentication. Under the Configuration Structure, select **Server Configuration** and click **Add**.

5. Start typing **Basic**, as shown in Figure 6-7 on page 188, select **Basic User Registry**, and click **OK**.

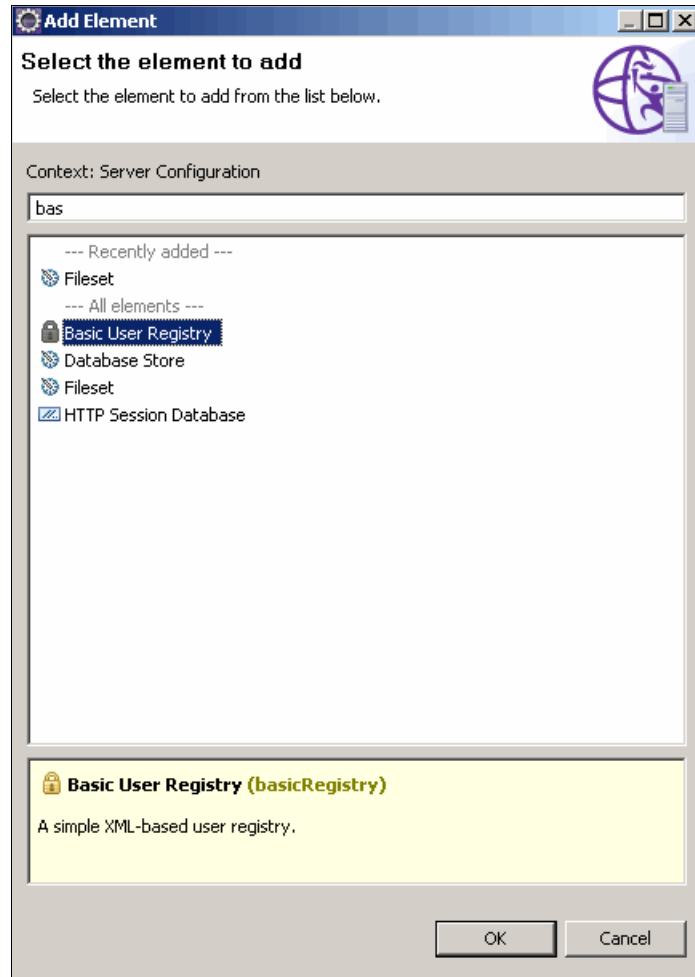


Figure 6-7 Adding Basic User Registry

6. Select **Basic User Registry** in the Configuration Structure. Under the Basic User Registry Details section to the right, enter an ID and Realm name for your registry, as shown in Figure 6-8.

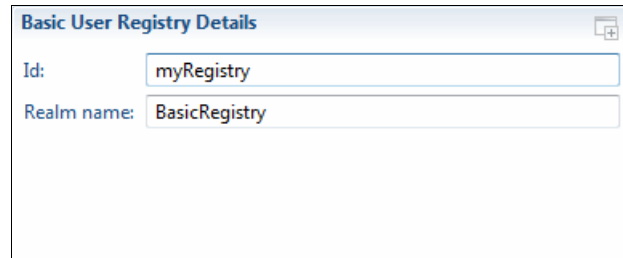


Figure 6-8 Entering the Basic Registry details

7. Now that you have a basic user registry, you must add users or groups to it. Therefore, click **Add**.
8. In the resulting window, select **User** and click **OK**.

9. A user object now appears under your basic user registry. Click the **User** object. Located to the right, under User Details, enter a user name of User1 and then click **Set** to enter a password, as shown in Figure 6-9. For the password encoding, you can use XOR encoding, AES encryption, or one-way hash. See the following link for more information about using encryption:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_pwd_encrypt.html

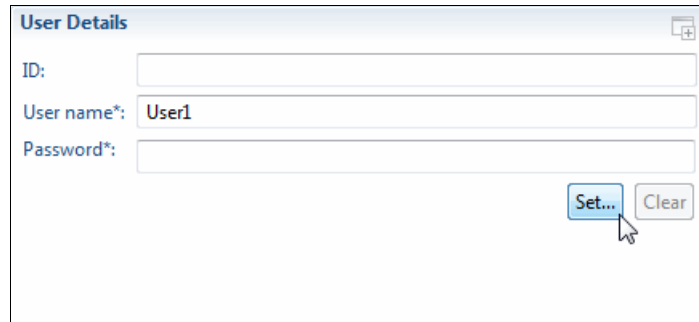


Figure 6-9 Adding a user

10. Enter the password and click **OK**. You have now created a user for the basic registry. Repeat the process and add another user that is called User2.
11. Then, add a security group to the basic user registry. Select the **Basic User Registry** and click **Add**.
12. Select **Group** from the Add Item window and click **OK**.
13. A group element appears under your basic user registry. Ensure that it is selected and under Group Details. Enter Group1 as the group name.
14. We must now add users to the group. Select the group under the Configuration Structure and click **Add**. In the Add Item window, select **member** and click **OK**.
15. Select the member element that appears under the group. In the Group Member Details window, enter User1 for the User name.
16. Save the changes. Your configuration Structure should now contain the basic user registry structure, as shown in Figure 6-10.

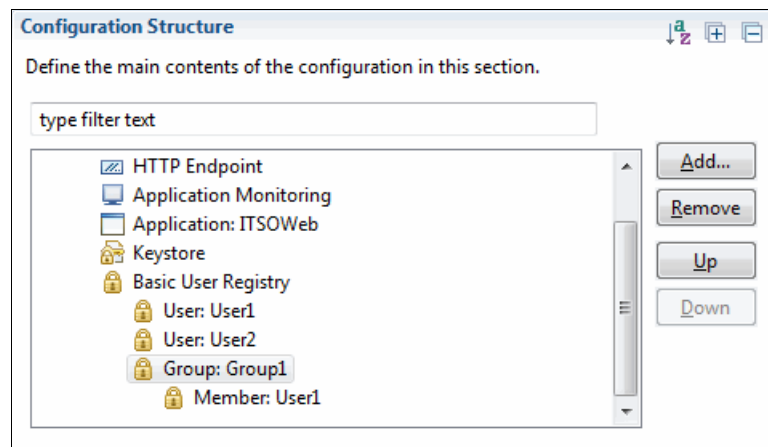


Figure 6-10 The completed basic user registry

6.3.2 Updating an application to support Form Login

To use Form Login for your application, you must make a number of changes to the application and its configuration. These changes are needed to allow the user to authenticate. The following sections describe the changes in detail.

Adding the login page

The user of the application must provide their login credentials to be authenticated. To allow the login process, provide a login page. This login page must contain a form that requests the user name and password and must always have the `j_security_check` action.

Example 6-3 shows an example form from the `login.html` page.

Example 6-3 The login form

```
<form method="POST" action="j_security_check">
<strong> Enter user ID and password: </strong>
<BR>
<strong> User ID</strong> <input type="text" size="20" name="j_username">
<strong> Password </strong> <input type="password" size="20" name="j_password">
<BR>
<strong> Click submit to authenticate: </strong>
<input type="submit" name="login" value="Login">
</form>
```

Use the `j_username` input field to obtain the user name and use the `j_password` input field to obtain the user password. When a user requests a web page that requires authentication, the web server stores the original request and displays the login form. When the login form is completed and the user is successfully authenticated, the server redirects the call to the original request.

Adding a login error page

If the authentication of a user fails, you must be able to advise the user of the failure. This is done by adding a login error page to your application. Example 6-4 shows an example of a `LoginError.html` file.

Example 6-4 An example login error page

```
<!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>Login Error</title>
</head>
<body>
<H1><B>A Form login authentication failure occurred</B></H1>
<P>Authentication might fail for one of many reasons. Some possibilities include:
<OL>
<LI>The user ID or password might have been entered incorrectly; either misspelled
or the wrong case was used.
<LI>The user ID or password does not exist, has expired, or has been disabled.
</OL>
</body>
</html>
```

Adding a form logout page

Form logout is a mechanism to log out without having to close all web browser sessions. After logging out, access to a protected web resource requires reauthentication.

Caution: Logout does not work for basic authentication (such as a browser pop-up window), as the browser stores user credentials in the HTTP headers and resends them on every request. You have to close all browser windows to successfully log out of the site protected by basic authentication.

The form logout page is an HTML or JSP file that you include with your web application. This logout page contains a form with a special post action. Example 6-5 provides an example of a `logout.jsp` file.

Example 6-5 A Sample logout page

```
<%@ 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>Insert title here</title>
</head>
<body>
<h2>Sample Form Logout</h2>
<form method="POST" action="ibm_security_logout" name="logout">
<BR>
<strong> Click this button to log out: </strong>
<input type="submit" name="logout" value="Logout">
<input type="HIDDEN" name="logoutExitPage" value="/login.html">
</form>
</body>
</html>
```

The `logoutExitPage` specifies where the user is to be redirected after logging out.

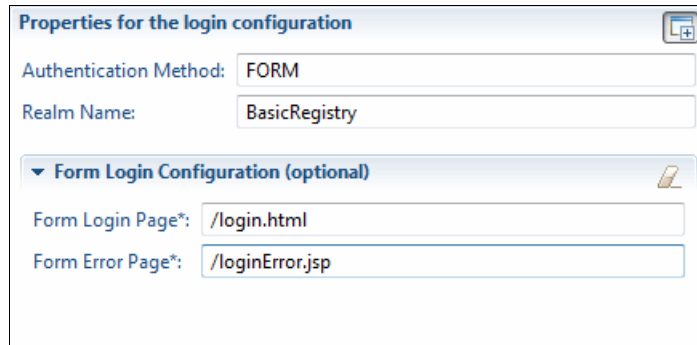
Configuring the application

The final changes that are needed to the web application are in the configuration. The web application must define the login mechanism and also restrict access to the relevant endpoints.

Although you do not need to do the HTTPS redirect as part of the form login, it is advised. Connection over HTTPS ensures confidentiality of the login credentials. The following steps expand upon the configuration you set up in 6.2, “HTTPS redirect” on page 185 to define the login mechanism and limit the access to the relevant endpoints:

1. In the Project Explorer, expand your web application and double-click **deployment descriptor**.
2. The first configuration that must be added is a login configuration (which defines the login mechanism). In the Overview section of the Deployment Descriptor Editor, select **Web Application** and click **Add**.
3. Select **Login Configuration** from the list and click **OK**.

4. Click the login configuration. You can now define how the user logs in by updating the properties for the login configuration, as shown in Figure 6-11.



Properties for the login configuration

Authentication Method: FORM

Realm Name: BasicRegistry

▼ Form Login Configuration (optional)

Form Login Page*: /login.html

Form Error Page*: /loginError.jsp

Figure 6-11 Defining the login configuration settings

Realm name: The realm name is an optional attribute that is used for differentiating between different repositories. If you have multiple repositories in your server, each one can have a realm name. You can then use the realm name to identify the repository to use for authentication. If you have only one repository, you can leave this value blank.

5. You now must restrict your endpoint to allow only certain users to access the resource. This is done by adding a security constraint and adding an authorization constraint to that security constraint. You already have a security constraint that you added in 6.2, “HTTPS redirect” on page 185. Use this constraint and add the authorization constraint to it. Select the **Security Constraint** from within the Overview section, as shown in Figure 6-12.

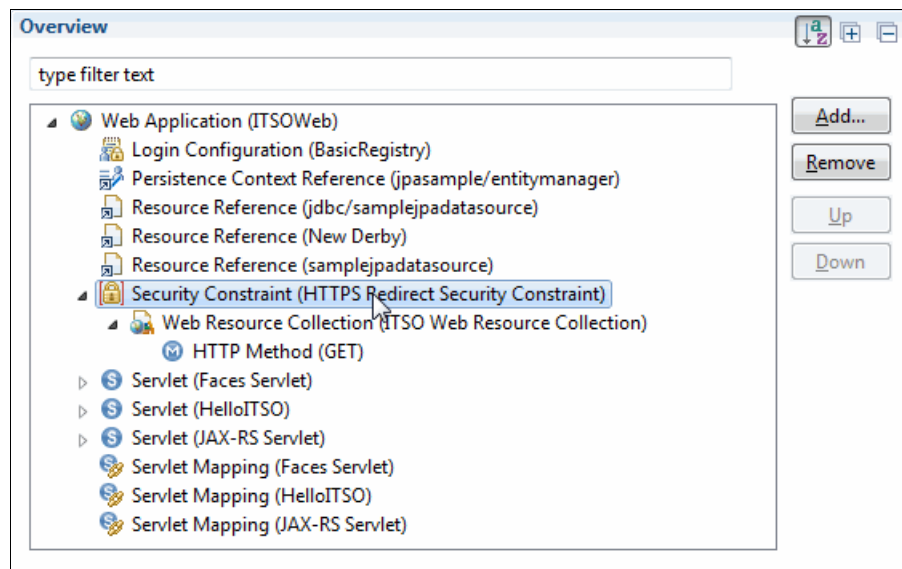


Figure 6-12 Selecting the Security Constraint

6. In the details section, under Authorization Constraint, click **Add** (located to the right of the Role Name). This adds a role that is required for access to the resource. Give the role a name, as shown in Figure 6-13.

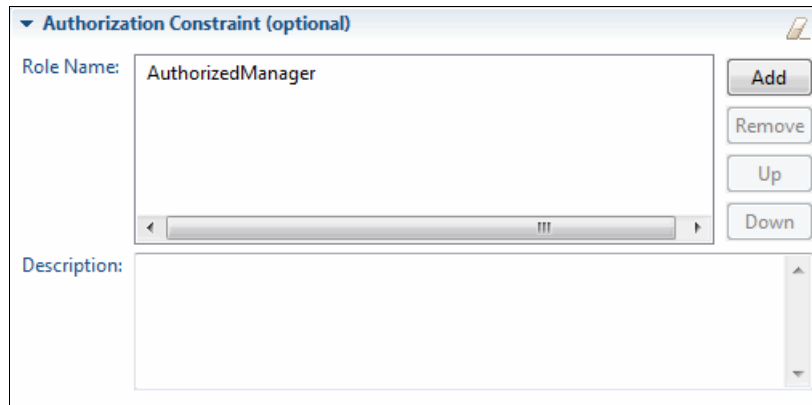


Figure 6-13 Adding an authorization role

Note: The Role Name within your application is a logical name and can be any value that you like. In 6.3.3, “Defining bindings between the application and the server” on page 193, you add mappings between your application roles and the users and groups that are defined in your user registry.

7. The application changes are now complete. Save all the changes.

6.3.3 Defining bindings between the application and the server

The final step in enabling Form Login is to map the roles that are defined within the application to the groups and users in your user registry. To do this, you must add an application binding in your server configuration using the following steps:

1. Open the Server view in Eclipse and double-click **Server Configuration** for your server. This opens your server.xml file. Switch to the design tab if it is not already selected.

2. In the Configuration Structure, select your application and click **Add**, as shown in Figure 6-14 on page 194.

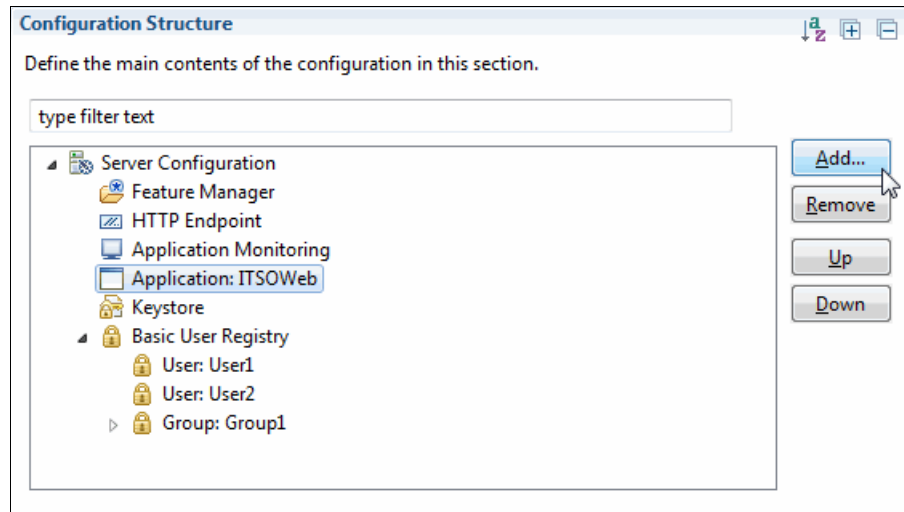


Figure 6-14 Adding an application binding

3. Select **Application Binding** from the list and click **OK**.
4. We now must specify the role to map. This should be the role that you defined in your application. Select **Application Binding** under the Configuration Structure and click **Add**.
5. Select **security-role** from the list and click **OK**.
6. Make sure that the security role is selected in the Configuration Structure and complete the security role name, as shown in Figure 6-15.

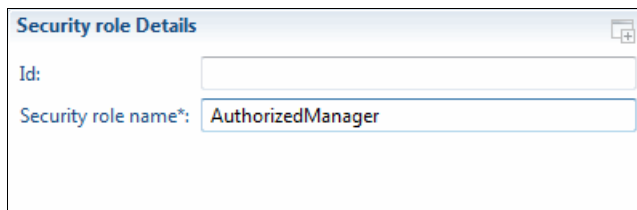


Figure 6-15 Defining the security role name

7. Map the security role with the users and groups that are to be considered part of this role. In our example, we are going to map Group1 to the AuthorizedManager role. Select the security-role under the Configuration Structure and click **Add**.
8. Select **group** from the list and click **OK**.

Note: As well as user and group, you can also select the special-subject option to assign access to either everyone or all.

9. Select the **Group element** in the Configuration Structure. Under Group Details, enter the name of the group that is to be allowed access to the resource, as shown in Figure 6-16.

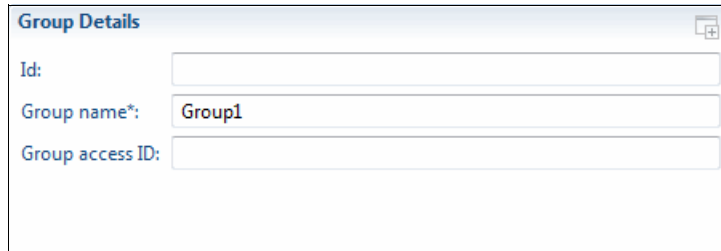


Figure 6-16 Entering the group details

10. Your Forms Login configuration is now complete. Save all changes.

If you now access your application, you are taken to the login window. When you enter the user credentials, only User1, which is in Group1, has access to the application.

6.4 Securing EJB applications

WebSphere Liberty also supports security in EJB components. You can define the method permissions using `ejb-jar.xml` or with annotations. Example 6-6 shows how to define security roles and method permissions using annotations. The `@DeclareRoles` annotation defines security role references. The `@RolesAllowed` annotation defines which roles are allowed to call a given method. You can also use `SessionContext` interface to obtain the user principal or check if the user has a certain role.

Example 6-6 Defining roles and method permissions via annotations

```
@Stateless
@LocalBean
@DeclareRoles("user")
public class SecuredEJB {
    @Resource
    private SessionContext ctx;

    @RolesAllowed("user")
    public String protectedMethod() {
        return " Method called by " + ctx.getCallerPrincipal();
    }
}
```

For more details about EJB security, see the following web site:

http://www-01.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wdt.doc/topics/csecuringjee.htm?cp=SSEQTP_8.5.5&lang=en

You can also find a lot of useful information about protecting EJB applications in Chapter 9, Securing an Enterprise JavaBeans application, in IBM Redbooks publication, *WebSphere Application Server V7.0 Security Guide*, SG24-7660. Although that publication is about the 7.0 version, most of the information presented there is also relevant to WebSphere Liberty.

6.5 Securing JMS applications

By default, JMS resources that are configured in the WebSphere Liberty profile server can be accessed by any user. You can enable JMS security to ensure that only authenticated and authorized users can access JMS resources, such as queues and topics.

You can enable JMS security by manually editing the list of features in the server configuration file to add `wasJmsSecurity-1.0`. You can also enable the feature in the WebSphere developer tools by loading the server configuration editor and adding the `wasJmsSecurity-1.0` feature to the feature manager.

After you enable the JMS security feature, you must authenticate with an authorized user to access any JMS resources.

6.5.1 Setting up user authentication

The first step to authenticating users is to create a user registry. Instructions for creating a basic user registry can be found in 6.3.1, “Defining the basic user registry” on page 187. You can also use a `quickStartSecurity` registry or an LDAP registry.

Users can be authenticated within application code or by configuration. The recommended approach is to authenticate using configured authentication data and avoid hardcoding authentication data in application.

To authenticate within the server configuration, you edit the properties for the JMS Connection Factory to specify authentication data. Use the following steps to complete that process:

1. Open the Server Configuration. Click **Add**, start typing **auth**, select **Authentication Data**, as shown in Figure 6-17 on page 197. Then, click **OK**.

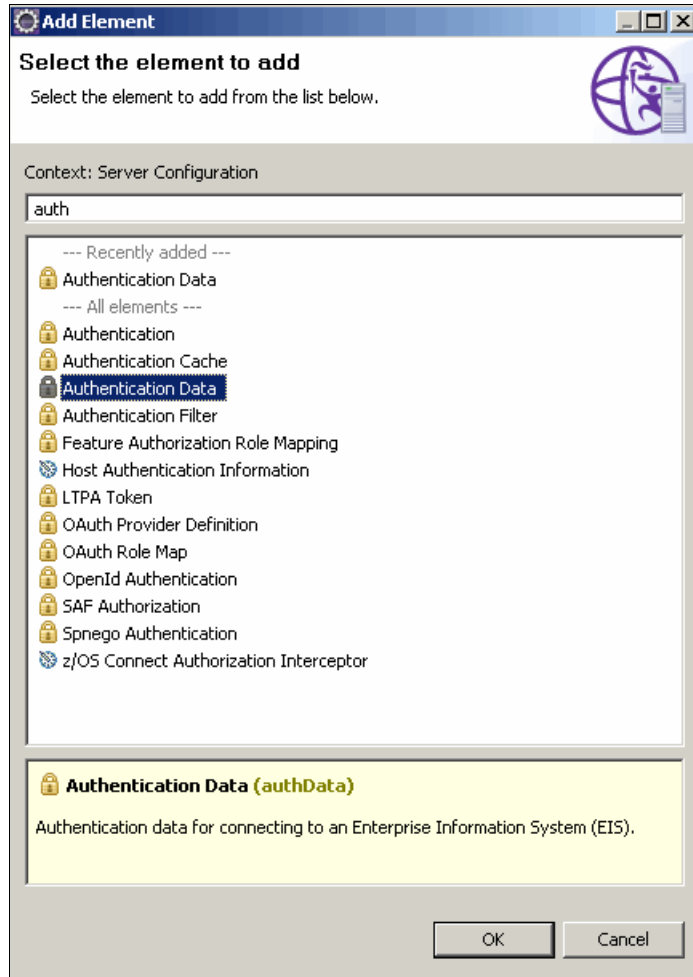


Figure 6-17 Adding authentication data

2. In the authentication data, provide the ID (for example `jmsUser`), username, and password.

3. Select **JMS Connection Factory**. In the Container managed authentication data reference, select the just added authentication data (jmsUser), as shown in Figure 6-18.

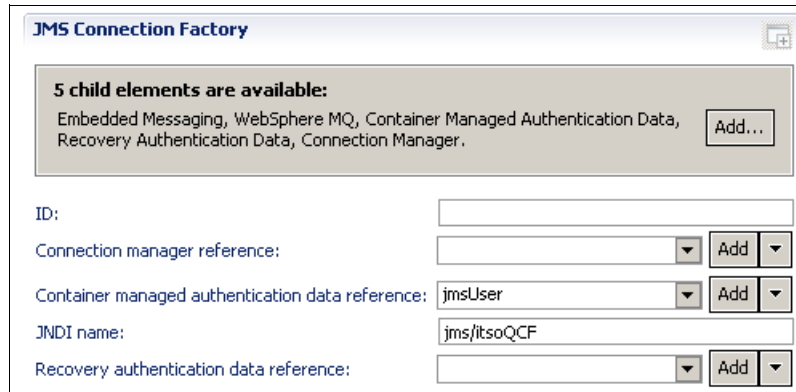


Figure 6-18 Configuring authentication reference

4. The relevant changes in the server.xml are shown in Example 6-7.

Example 6-7 Using authentication alias

```
<server description="new server">
  ....
  <authData user="user1" password="password" id="jmsUser"></authData>
  <jmsConnectionFactory jndiName="jms/itsoQCF" containerAuthDataRef="jmsUser">
    <properties.wasJms/>
  </jmsConnectionFactory>
</server>
```

You can also store the user name and password directly in the Embedded Messaging entry (represented as attributes of the `properties.wasJms` element) in the `server.xml` file, as shown in Example 6-8.

Example 6-8 Authenticating using `properties.wasJms`

```
<jmsConnectionFactory jndiName="jms/itsoQCF">
  <properties.wasJms password="password" userName="user1"/>
</jmsConnectionFactory>
```

To authenticate in the application code, provide the user name and password as parameters when you create a `JMSContext` in your application. Example 6-9 shows how you use the JMS API to authenticate.

Example 6-9 Authenticating within application code

```
@Resource(lookup="jms/itsoQCF")
private ConnectionFactory qcf;
....
jmsContext = qcf.createContext("user1", "password");
jmsContext.createProducer().send(queue, message);
```

When you have a message-driven bean in your application, you need to specify an authentication data reference in the activation specification (as shown in Example 6-10). You use the same authentication data that was configured earlier for the connection factory.

Example 6-10 Configuring authentication data in activation specification

```
<jmsActivationSpec authDataRef="jmsUser" id="ITSOJMSApp/SimpleMDB">
  <properties.wasJms destinationRef="jmsItsoQ"
destinationType="javax.jms.Queue"/>
</jmsActivationSpec>
```

For more information about authentication, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.n.d.multiplatform.doc/ae/twlp_msg_sec_authenticate.html?lang=en

6.5.2 Setting up user authorization

To access JMS resources, an authenticated user must be granted authorization to access the resources. Authorization can be configured by editing the server configuration to add roles, users, and groups.

To complete this process, use the following steps:

1. Set up authorization in the Liberty profile developer tools. Open the server configuration editor and select **Messaging Engine**. Click **Add** and select **Messaging Security**, as shown in Figure 6-19.

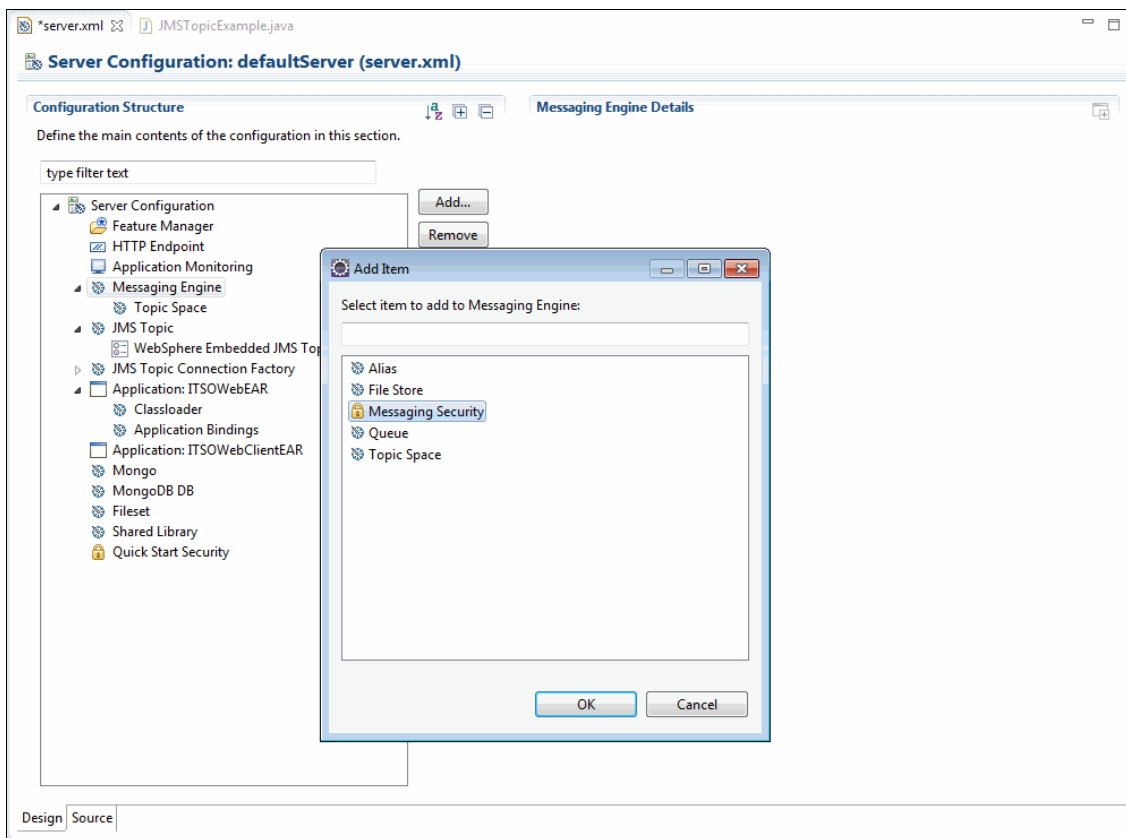


Figure 6-19 Adding Messaging Security

2. Now, select **Role** underneath Messaging Security. In the details window, enter a name (such as `developer`) in the Role name field. With **Role** still selected, click **Add**, to add a

user. In the User details window, enter a user name, such as User1, from your user registry.

3. Add permissions for the role by selecting **Role**, then clicking **Add**, and selecting **Queue Permission**. In the Action field of the details window, you can configure whether the user is authorized to send, receive, browse, or all. Select an action from the drop-down next to the Action field (such as **ALL**). You can add additional actions by using the Add button.

In the Queue reference field, enter a value that corresponds to the queue ID.

You can also manually edit configurations. Example 6-11 on page 200 shows the configuration that must be added to the server.xml file for manual editing.

Example 6-11 Configuration for manual editing

```
<essagingEngine>
  <queue id="itsoQ"/>
  <essagingSecurity>
    <role id="developer" name="developer">
      <user name="user1"/>
      <queuePermission queueRef="itsoQ">
        <action>ALL</action>
      </queuePermission>
    </role>
  </essagingSecurity>
</essagingEngine>
```

6.6 JAX-WS security

Web services in Liberty profile server can be secured either at the transport layer or at the messaging layer. Configuring transport security allows you to authenticate and ensure confidentiality of HTTP communications between a web service client and a web service provider. Transport security can be enabled using SSL, basic authentication, or client certificate authentication.

Message-level security allows you to secure SOAP messages that are exchanged between web service clients and web service providers using WS-Security standards. WebSphere Liberty profile supports the following web service standards:

- ▶ Web Services Security SOAP Message Security 1.1
- ▶ Web Services Security Username Token Profile 1.1
- ▶ Web Services Security X.509 Certificate Token Profile 1.1
- ▶ WS-SecurityPolicy 1.3

This section provides an example of how to secure a web service using SSL at the transport layer and Username Token authentication at the message layer. For more information about other ways to secure web services, see the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/cwlp_wssec.html?cp=SSEQTP_8.5.5%2F1-3-11-0-4-9

This example builds on the JAX-WS example in 3.1.6, “Developing JAX-WS web services applications with the Liberty profile developer tools” on page 78. Before you begin, make sure that you enabled SSL by creating a keystore, as described in 6.1, “Enabling SSL” on page 182. Also, make sure that the appSecurity-2.0 feature is enabled.

In addition to creating a keystore, you must create an SSL Default Repertoire. To complete that process, use the following steps:

1. In the Liberty profile developer tools server configuration editor, create the default repertoire by selecting the server configuration element.
2. Click **Add** and then select **SSL Default Repertoire**. In the details window, enter `defaultSSLConfig` in the Default SSL repertoire field. Outside of the tool environment, create the `sslDefault` element, as shown in Example 6-12.

Example 6-12 Default SSL Repertoire

```
<sslDefault sslRef="defaultSSLConfig"/>
```

3. You must also enable HTTPS redirect for the pattern `/ITS0GreeterService` so that requests to the web service over HTTP are redirected to the SSL port using HTTPS. To accomplish this task, use the procedure in 6.2, "HTTPS redirect" on page 185. After enabling redirect, test that it is working by visiting the following link:

`http://localhost:9080/ITS0Web/ITS0GreeterService?wsdl`

You should be redirected to the following ULR:

`https://localhost:9443/ITS0Web/ITS0GreeterService?wsdl`

You should see the contents of the WSDL document.

4. To enable the web services security run time, you must enable the `wsSecurity-1.1` feature in the server configuration.
5. To configure the JAX-WS run time to send a Username Token with web service client requests, you can add a `wsSecurityClient` element to the server configuration. When it is specified, the WS-Security client configuration becomes the default configuration for all web services clients on the server. The configuration might be overridden programmatically by passing in property values in the `RequestContext` in the client code. Example 6-13 shows an example WS-Security client configuration.

Example 6-13 WS-Security Client configuration

```
<wsSecurityClient ws-security.password="password1"  
ws-security.username="user1"/>
```

6. The WS-Security provider can also have a default configuration that is specified in the server configuration. To specify that web service providers should place the identity of the Username Token from the SOAP message on the thread, configure a `wsSecurityProvider` element in the `server.xml` file, as shown in Example 6-14.

Example 6-14 WS-Security Provider configuration

```
<wsSecurityProvider ws-security.username="user1">  
  <callerToken name="UsernameToken"/>  
</wsSecurityProvider>
```

7. You must also configure the WSDL for the web service provider and client to enforce SSL and to require a user name token. The Liberty profile developer tools make it easy to add web service security policy templates to the WSDL.

8. If you did not generate a WSDL file for the web service, you can do so now by following the procedure in “Creating a web service” on page 79 and making sure that the **Generate WSDL file into the project** option is selected in the second window. The wizard does not overwrite any existing artifacts without confirmation, so you do not lose any modifications that you made.
 - a. Begin by expanding **ITSOWeb** → **Services** and selecting (<http://example.itso.com/ITSOGreeterService>). Right-click the **service** and select **Add Security Policy to Service WSDL**. In the wizard, give the policy a name, such as **UsernameTokenSSL**. In the drop-down field for Policy template, select **UsernameToken with password text, nonce, and created timestamp over SSL**, as shown in Figure 6-20 on page 202.

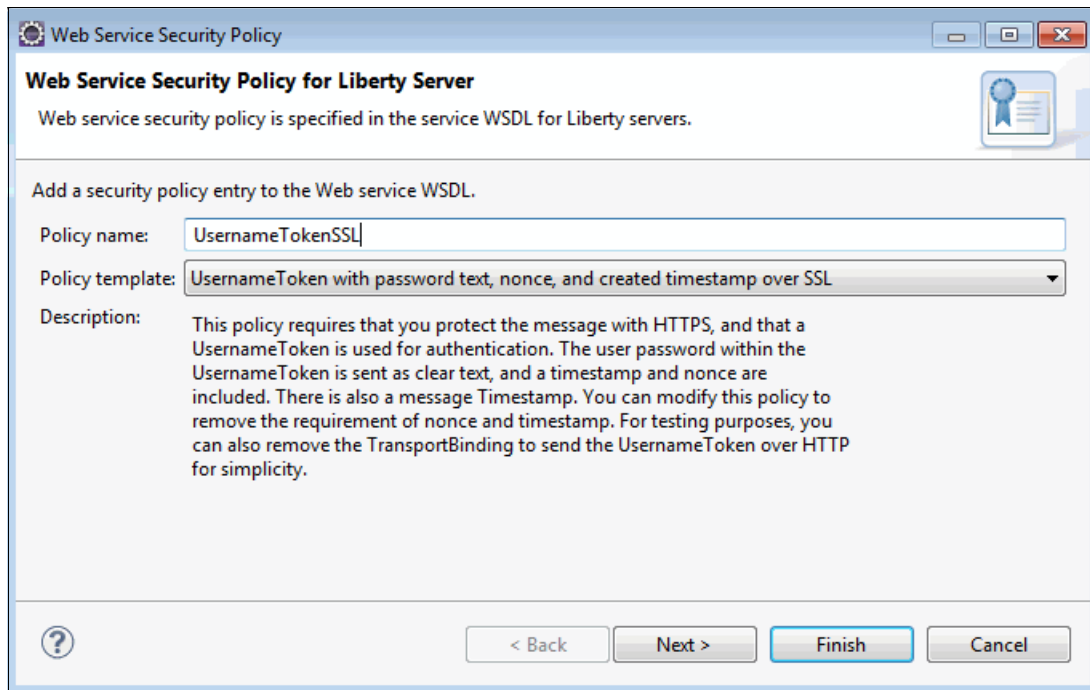


Figure 6-20 Web Service Security Policy wizard

- b. Click **Next** and then select the **check box** to attach the security policy to the *ITSOGreeterPortBinding*. Click **Finish** and open the WSDL document in *ITSOWeb/WebContent/WEB-INF/wsd1*. A Policy element is added with the name *UsernameTokenSSL*. Also, there is now a PolicyReference element located underneath the binding element. With these changes to the WSDL file, the web service now generates an error if a client attempts to access the service without SSL or without a user name token.

In the same WSDL file, we can ensure that SSL is used by changing the address for the *ITSOGreeterPort*. The new service definition is shown in Example 6-15.

Example 6-15 ITSOGreeterService definition

```

<service name="ITSOGreeterService">
  <port name="ITSOGreeterPort" binding="tns:ITSOGreeterPortBinding">
    <soap:address location="https://localhost:9443/ITSOWeb/ITSOGreeterService"/>
  </port>
</service>

```

9. You can attempt to test these changes now by visiting the link <http://localhost:9080/ITSOWebClient/JAXWSGreeterClient>. However, the test should fail because the client does not yet send a user name token.
10. Now, make the same changes that were made to the provider WSDL to add the WS-Security policy and update the address to the client WSDL in the ITSOWebClient project. After making these changes, you can try again visiting the test client. This time, you should see the output Hello, JAX-WS Client from the ITS0 Greeter that indicates that the service was successfully started.

6.7 Securing NoSQL applications

NoSQL databases are becoming more popular. However, most of them are not protected by default and allow anybody to perform all operations. For production deployments of applications using such databases, you should enable protection on the databases and access them in a secure way.

6.7.1 Securing MongoDB applications

Applications that use MongoDB can be secured by using either application-managed security or container-managed security. You must also enable access control on the MongoDB instance. For more information about enabling database authorization in MongoDB, see the following website:

<http://docs.mongodb.org/manual/core/security>

In application-managed security, you make changes inside the application code to authenticate with the MongoDB instance. The code in Example 6-16 shows how to use the MongoDB APIs to authenticate.

Example 6-16 Authenticating with a MongoDB instance

```
// mongoDB is an instance of com.mongodb.DB
if (!mongoDB.isAuthenticated())
    mongoDB.authenticate("user1", "password1".toCharArray());
```

In container-managed security, you edit the server configuration to specify a user name and password for the MongoDB instance. Example 6-17 shows how you specify the user name and password in the `server.xml` file.

Example 6-17 Configuring authentication in a server.xml file

```
<mongo id="mongoInstance1" libraryRef="mongoLibrary" user="user1"
password="password1"/>
```

6.7.2 Securing CouchDB applications

CouchDB databases are unprotected by default and everyone has privileges to do anything. For more information about enabling database security in CouchDB, see the following website:

<http://docs.couchdb.org/en/latest/intro/security.html>

You can specify user name and password for a user accessing CouchDB in the Server configuration. To Configure the CouchDB element as shown in Figure 6-21 on page 204, double-click **Server configuration** in the Servers view, and select **CouchDB element** in the Server Configuration. Enter the username in the CouchDB user ID field and password in the password field.

Figure 6-21 Configuring CouchDB

You can also edit the server.xml file manually. Add the username and password elements, which are shown in Example 6-18.

Example 6-18 CouchDB element in server.xml

```
<couchdb username="admin" password="admin" libraryRef="couchLibrary"
jndiName="couchdb/connector" url="http://localhost:5984" id="couchdb"></couchdb>
```

6.8 Authenticating users in Liberty profile

WebSphere Liberty profile server uses a user registry to get information about users and groups. The user registry is also used during authentication and authorization process.

For more information about how authentication is handled in Liberty, see the following page:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_authentication.html?lang=en

6.8.1 User registries for WebSphere Liberty profile

The user registries store user and group information that is used for authentication purposes. Liberty profile supports the following types of user registries:

- ▶ Basic user registry: A simple registry, with all information contained in the server.xml file, mostly for development and prototyping purposes.
- ▶ LDAP registry: An extensive LDAP registry, supporting federation of multiple LDAPs and failover. Recommended for production deployments.
- ▶ Custom registry: You can develop a custom user registry class by implementing the com.ibm.websphere.security.UserRegistry and deploy as Open Service Gateway initiative (OSGi) service.

- ▶ SAF registry: On the z/OS platform, you can configure Liberty to use the SAF registry.

For more information about configuration of these registries, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_sec_registries.html

6.8.2 Custom authentication methods

In this section, various authentication options are briefly described.

Using a servlet API

Java EE servlet API provides methods that can be used while customizing the authentication process. The most useful methods from the `HttpServletRequest` interface are noted in the following list:

- ▶ `login(String username, String password)`: Authenticates user to the server with given user ID and password. If authentication is successful, it creates subject and LTPA cookie.
- ▶ `logout()`: Logs out the current user. This method invalidates user's HTTP session, clears LTPA cookie, and removes user from the authentication cache.
- ▶ `authenticate(HttpServletRequest response)`: Invokes authentication mechanism currently configured in given application. Method returns true, if authentication was successful.

For more information about these methods, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.base.doc/ae/tsec_web.html?cp=SSEQTP_8.5.5%2F1-11-2-7-2-1&lang=en

Using a Trust Association Interceptor (TAI)

Sometimes, you need to provide a custom authentication process with additional parameters required or integrate authentication with a third-party security service. In such scenarios, you can develop TAI. To create TAI, you need to implement the `com.ibm.wsspi.security.tai.TrustAssociationInterceptor` interface.

Once TAI is implemented, you need to add it to the Liberty profile server. You can use one of the following methods:

- ▶ Put the custom TAI class in a JAR file, for example `simpleTAI.jar`; then, make the JAR file available as a shared library. See the following website for Configuring TAI for the Liberty profile page:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_sec_tai.html

- ▶ Package the custom TAI class as a feature. See the following website for Developing a custom TAI as a Liberty profile feature page:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_feat_tai.html

For more information about TAI in general, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_dev_custom_tai.html

Using a login module

WebSphere Liberty profile provides Java Authentication and Authorization Service (JAAS) plug-in points to modify system login configuration. Login modules are best used when the authentication process suits your needs in general but you would like to add custom information to the created Subject.

To create a login module, you need to implement `javax.security.auth.spi.LoginModule` interface and then add it to the `system.WEB_INBOUND` login configuration.

For more information about developing login modules, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_dev_custom_jaas.html

Using OpenID

OpenID is an open standard that allows users to authenticate themselves to multiple sites without the need to manage multiple accounts. WebSphere Liberty profile supports OpenID 2.0 and plays a role as a Relying Party in web single-sign-on.

The user accessing a protected application is redirected to the OpenID provider site for credential verification and then redirected back to the original site with authentication result.

For more information about using OpenID, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_openid.html

Using OpenID Connect

OpenID Connect is another standard that is used by web single-sign-on. It is built on top of the OAuth 2.0 protocol. This protocol enables client applications to rely on authentication that is performed by an OpenID Connect Provider to verify the identity of a user. WebSphere Liberty profile supports OpenID Connect 1.0 and can act as Client, Relying Party, and Provider.

For example, this mechanism can be used to authenticate users to your application using their Google credentials. See the following website for more information about this topic:

<https://www.youtube.com/watch?v=Rfxy0FK0fgw>

For more information about using OpenID Connect, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_openid_connect.html

For more information about OAuth support, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_oauth_introduction.html?lang=en-us

Using JASPIC

The Java Authentication SPI for Containers specification (JASPIC) defines a standard way to plug in your custom authentication provider. You can develop your JASPIC provider as a Liberty user feature or embed the provider in your application.

For more information about developing using JASPIC, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_develop_jaspic.html

Using SPNEGO

You can use single sign-on for HTTP requests by using the Simple and Protected GSS API Negotiation Mechanism (SPNEGO) web authentication for WebSphere Liberty profile. SPNEGO single sign-on enables HTTP users to log in to a Microsoft domain controller only once at their desktop and to achieve single sign-on (SSO) with the Liberty profile server.

Note: SPNEGO is only supported in WebSphere Liberty profile with the IBM JDK.

For more information about configuring SPNEGO in Liberty, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_spnego_config.html

Authentication filters

Authentication filters let you configure which requests should be processed by certain authentication providers (such as OpenID, OpenID Connect, or SPNEGO).

In the provider configuration, you define reference to the authentication filter. If all conditions defined in the filter are met, the request is processed by that provider.

Authentication filter can use any of the following elements: `userAgent`, `host`, `webApp`, `remoteAddress`, and `requestUrl`.

For more information about how to configure authentication filter, see the following website:

https://infocenters.hursley.ibm.com/wlpsolo/v85/draft/help/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_auth_filter.html



Serviceability and troubleshooting

The Liberty profile has several tools to help identify problems with the server and the applications that are deployed to it.

This chapter contains the following sections:

- ▶ Logs and trace
- ▶ Server memory dump
- ▶ MBeans and JConsole
- ▶ OSGi Debug console
- ▶ Event logging feature
- ▶ Slow request detection and hung request detection capabilities

7.1 Logs and trace

The Liberty profile can produce various traces that enable you to debug issues with the server and your applications. By default, no tracing is enabled. However, you can enable tracing if required. In this section, we describe where to find the output and how to configure what trace data is collected.

7.1.1 Inspecting the output logs and trace

The Liberty profile server records limited information by default. This basic information is useful for debugging common configuration issues. You can view the output logs by opening the `${server.output.dir}/logs/messages.log` file. Example 7-1 shows a sample output from a server start.

Example 7-1 Sample content of the message.log file

```
[5/19/15 14:22:41:589 EDT] 00000707 com.ibm.ws.app.manager.AppMessageHelper
I CWWKZ0018I: Starting application ITSOWeb.
[5/19/15 14:22:41:691 EDT] 00000707 com.ibm.ws.webcontainer.osgi.webapp.WebGroup
I SRVE0169I: Loading Web Module: ITSOWeb.
[5/19/15 14:22:41:692 EDT] 00000707 com.ibm.ws.webcontainer
I SRVE0250I: Web Module ITSOWeb has been bound to default_host.
[5/19/15 14:22:41:692 EDT] 00000707 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host): http://localhost:9087/ITSOWeb/
[5/19/15 14:22:41:694 EDT] 00000707 com.ibm.ws.app.manager.AppMessageHelper
A CWWKZ0003I: The application ITSOWeb updated in 0.104 seconds.
[5/19/15 14:23:29:560 EDT] 00000705 com.ibm.ws.session.WASSessionCore
I SESN0176I: A new session context will be created for application key default_host/ITSOWeb
[5/19/15 14:23:29:560 EDT] 00000705 com.ibm.ws.util
I SESN0172I: The session manager is using the Java default SecureRandom implementation for
session ID generation.
[5/19/15 14:23:29:568 EDT] 00000705 com.ibm.ws.jsp
I JSPG8502I: The value of the JSP attribute jdkSourceLevel is "15".
[5/19/15 14:23:29:621 EDT] 00000705
org.apache.myfaces.config.DefaultFacesConfigurationProvider I Reading standard config
META-INF/standard-faces-config.xml
[5/19/15 14:23:29:644 EDT] 00000705
org.apache.myfaces.config.DefaultFacesConfigurationProvider I Reading config
/WEB-INF/faces-config.xml
[5/19/15 14:23:29:708 EDT] 00000705 org.apache.myfaces.application.ApplicationImpl
I Couldn't discover the current project stage, using Production
[5/19/15 14:23:29:708 EDT] 00000705 org.apache.myfaces.config.FacesConfigurator
I Serialization provider : class
org.apache.myfaces.shared_impl.util.serial.DefaultSerialFactory
[5/19/15 14:23:29:708 EDT] 00000705 org.apache.myfaces.webapp.AbstractFacesInitializer
I ServletContext 'C:\IBM\ch3_ITSOWeb\ITSOWeb\WebContent' initialized.
[5/19/15 14:23:29:708 EDT] 00000705 com.ibm.ws.webcontainer.servlet
I SRVE0242I: [ITSOWeb] [/ITSOWeb] [Faces Servlet]: Initialization successful.
[5/19/15 14:23:29:724 EDT] 00000705
apache.wink.server.internal.application.ApplicationProcessor I The following JAX-RS
application has been processed:
com.ibm.ws.jaxrs.webcontainer.JAXRSDefaultApplicationSubclassProxy
[5/19/15 14:23:29:724 EDT] 00000705 org.apache.wink.server.internal.log.Resources
I The server has registered the JAX-RS resource class com.itso.example.ITSOJaxrsExample
with @Path(/example).
[5/19/15 14:23:29:724 EDT] 00000705 org.apache.wink.server.internal.log.Providers
I There are no custom JAX-RS providers defined in the application.
```

```
[5/19/15 14:23:29:806 EDT] 00000705 com.ibm.ws.webcontainer.servlet
I SRVE0242I: [ITSOWeb] [/ITSOWeb] [JAX-RS Servlet]: Initialization successful.
```

The log shows the server start procedure. First, the kernel starts. Then, the feature manager initializes and reads the configuration from the `server.xml` file. The server is configured to listen on a given port. Then, the ITSOWeb application is started and finally the server is ready to serve the application.

7.1.2 Configuration of an additional trace

A typical task when working with applications is to enable and configure the logging properties of the runtime server to inspect for problems. The Liberty profile server can be configured to gather debug information for both runtime issues and application code.

The Liberty profile server, when started, is configured to capture a minimal amount of trace information. You can modify this default by specifying properties in the server configuration file or `bootstrap.properties` file. Open Service Gateway initiative (OSGi) logging output is intercepted and delivered through the trace support. There is also interception of `java.util.logging` output.

To learn more about configuring the logging service, see the IBM Knowledge Center at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.0.0/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html

Configuring trace in the servers configuration

To configure an additional trace in the servers configuration, follow these steps:

1. Open the **Servers** view in your workbench and double-click **Server Configuration** for your server. This opens your `server.xml` file. Select the **Design** tab.
2. Under the Configuration Structure, select **Server Configuration** and click **Add**, as shown in Figure 7-1.

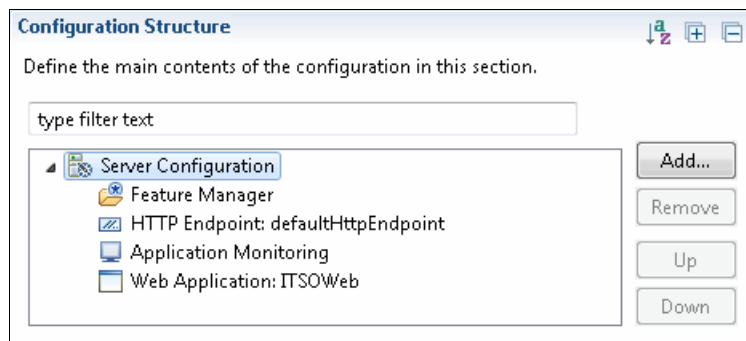
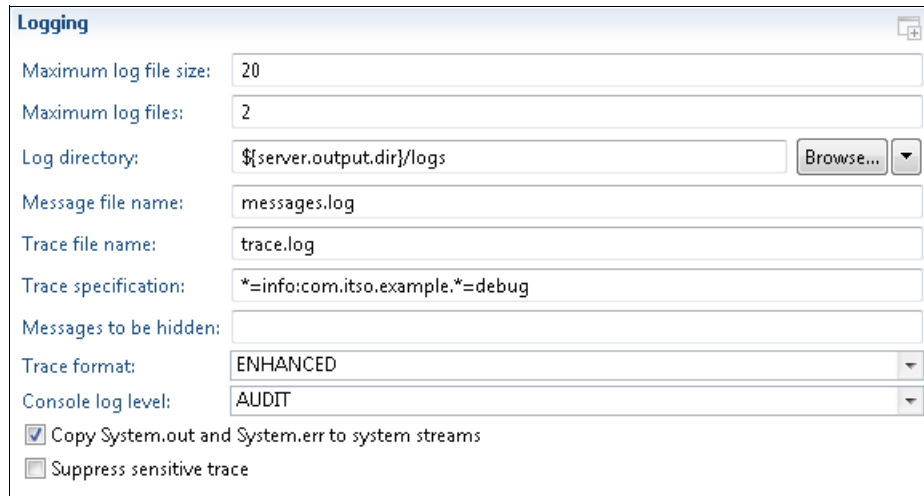


Figure 7-1 Adding the logging option to the server configuration

3. Select **Logging** and click **OK**.

4. Ensure that logging is selected under the Configuration Structure. You can now configure logging in the Logging details section, as shown in Figure 7-2.



The screenshot shows a 'Logging' configuration window with the following settings:

- Maximum log file size: 20
- Maximum log files: 2
- Log directory: `${server.output.dir}/logs` (with a 'Browse...' button and a dropdown arrow)
- Message file name: `messages.log`
- Trace file name: `trace.log`
- Trace specification: `*=info:com.itso.example.*=debug`
- Messages to be hidden: (empty)
- Trace format: ENHANCED (dropdown menu)
- Console log level: AUDIT (dropdown menu)
- Copy System.out and System.err to system streams
- Suppress sensitive trace

Figure 7-2 Entering the trace details

The Logging details window allows you to configure many aspects of logging, including the location of the logs and where they are stored. The Trace Specification defines what information to capture in the logs. In our example, we capture information-level messages for all classes. In addition, we chose to capture all debug or higher-level messages for any `com.itso.example.*` classes.

The `trace.log` file is located by default in the `${server.output.dir}/logs/` directory.

For information about how to add logging to your applications, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_javalogapps.html

Configuring trace using bootstrap.properties

Using the server configuration file to enable logging is an easy and efficient method. However, the following list notes some cases where you might want to put logging properties in to the `bootstrap.properties` file:

- ▶ When logging is required before the configuration processing occurs at server start (this might be required for IBM support for debugging a server start problem).
- ▶ When the `server.xml` configuration is shared by multiple server instances, and logging is only required on one server (then the `bootstrap.properties` file for that server can be modified).

Example 7-2 shows a sample `bootstrap.properties` file that enables tracing of any `com.itso.example.*` classes.

Example 7-2 A sample `bootstrap.properties` file

```
# New trace messages are written to the end of the file (true or false)
com.ibm.ws.logging.trace.append = "true"
# maximum size of each trace file (in MB)
com.ibm.ws.logging.trace.max.file.size = 2
# maximum number of trace files
com.ibm.ws.logging.trace.max.files = 2
# Trace format (basic or advanced)
com.ibm.ws.logging.trace.format = "basic"
# Trace settings string - this string enables all trace
com.ibm.ws.logging.trace.specification = com.itso.example.*=debug
```

7.2 Server memory dump

To capture state information of a Liberty profile server, you can use the **dump** command. This can be useful for problem diagnosis of a Liberty profile server.

The directory of results that is generated by the **dump** command contains server configuration, log information, and details of the deployed applications in the `workarea` directory. The **dump** command can be used against a running or stopped server. If the server is running, the following additional information is gathered:

- ▶ State of each OSGi bundle in the server
- ▶ Wiring information for each OSGi bundle in the server
- ▶ Component list managed by the Service Component Runtime (SCR)
- ▶ Detailed information of each component from SCR
- ▶ The current state of all requests still running in the system (if the `requestTiming-1.0` feature is enabled)
- ▶ The statistics about each of the JDBC requests the server has tracked (if `timedOperations-1.0` feature is enabled)

The following syntax is an example of running the **dump** command:

```
server dump server1 --archive=c:\tmp\server1_dump.zip
```

You can also create a memory dump of the Liberty profile server by using the WebSphere developer tools. This is done by right-clicking the server in the Servers view and selecting **Utilities** → **Generate Dump for Support**. You have the option to generate a server or a JVM dump. The server dump can contain heap, system, or thread information. The JVM dump can contain heap or system information.

7.3 MBeans and JConsole

To monitor the Liberty profile server runtime characteristics, administrators and developers can use standard tools for Java runtimes, such as the JConsole.

The Liberty profile offers the `monitor-1.0` feature. This feature allows users to track information about the Liberty profile server runtime, such as JVM, web applications, and the thread pool. To enable this monitoring, add the `monitor-1.0` feature to the `server.xml` configuration so that the following MBeans are available:

- ▶ `WebSphere:type=JvmStats`
- ▶ `WebSphere:type=ServletStats,name=*`
- ▶ `WebSphere:type=ThreadPoolStats,name=Default Executor`
- ▶ `[8.5.5.0 or later]org.apache.cxf:type=WebServiceStats,service=*,port=*`
- ▶ `[8.5.5.0 or later]WebSphere:type=SessionStats,name=*`
- ▶ `[8.5.5.0 or later]WebSphere:type=ConnectionPool,name=*`

For more information about monitoring the Liberty profile using MBeans, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_mon.html

JConsole is shipped as part of the Java Runtime package. To run JConsole against the Liberty profile, start it from the command line, as shown in the following syntax:

```
C:\Java\jdk1.7.0_67\bin>jconsole.exe
```

When JConsole launches, a welcome page opens. You must connect to the Java process that runs the Liberty profile. The connection is made by logging in. The JConsole process must run under the same operating system user ID and use the same Java runtime as the server. As illustrated in Figure 7-3, we used the local server for our example.

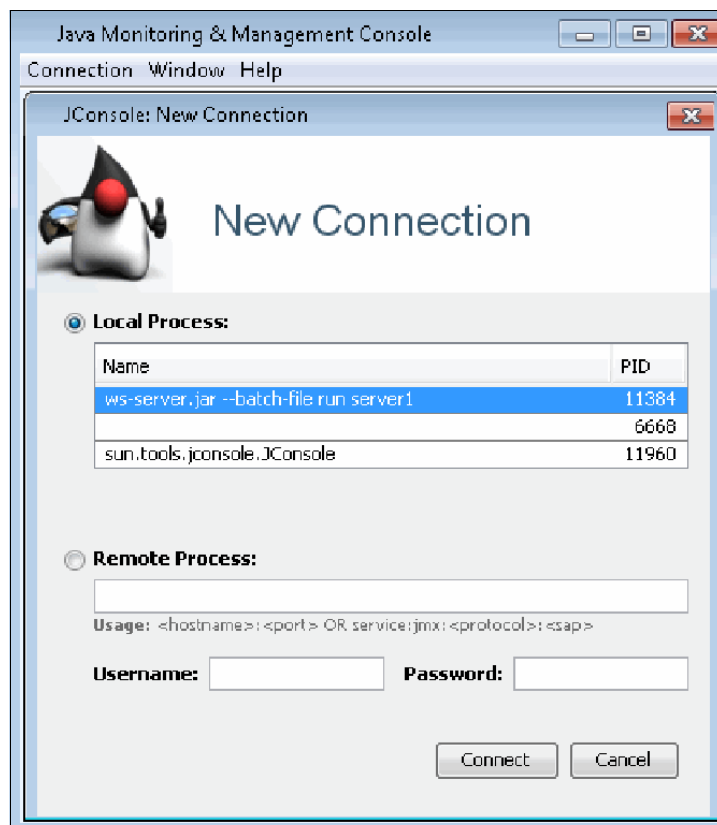


Figure 7-3 Establishing a connection using JConsole

The JConsole loads all runtime information into the tool. This includes information about the Java runtime, heap size, or processor usage, as illustrated in Figure 7-4.

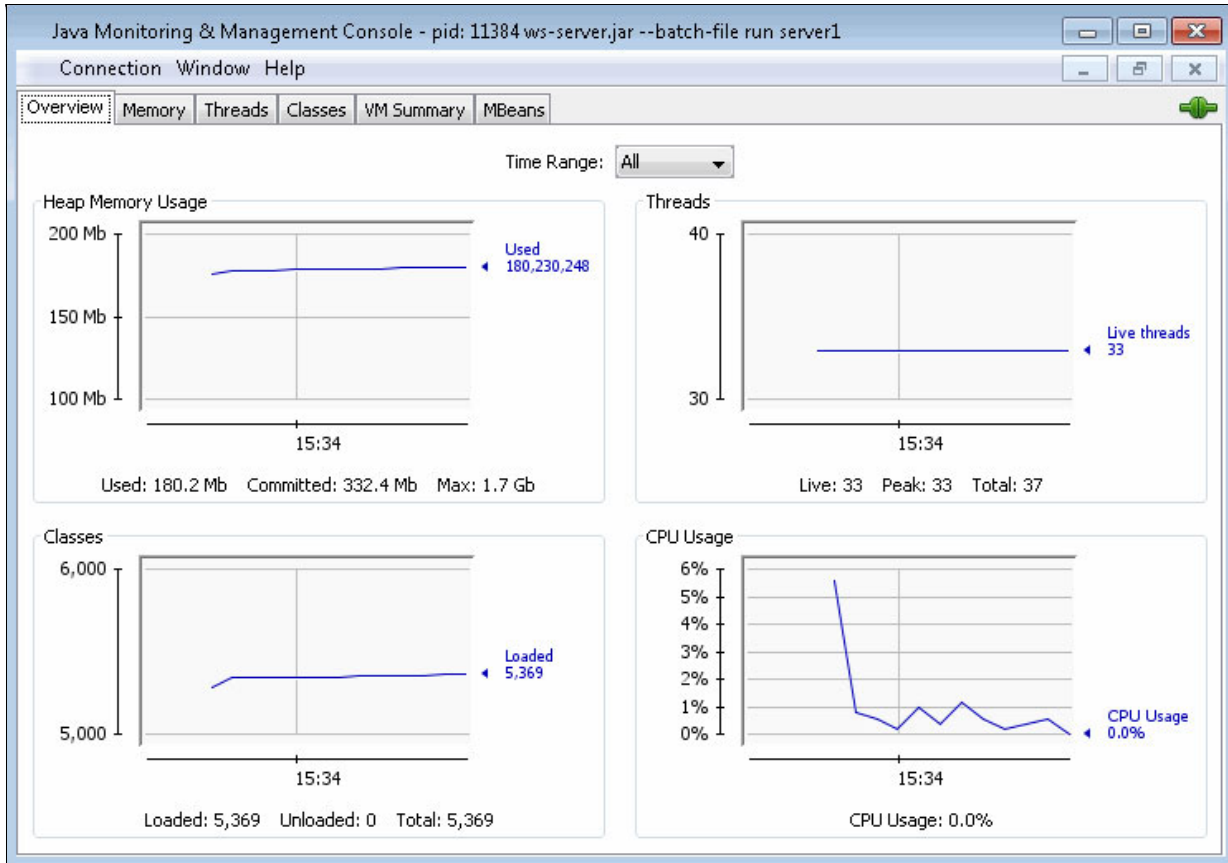


Figure 7-4 Example of monitoring the output information using JConsole

Using the JConsole tool, you can also trigger MBean operations that are part of the Liberty profile. MBeans are available under the MBeans tab. Figure 7-5 shows an example of issuing a **restart** command on the ITSOWeb application.

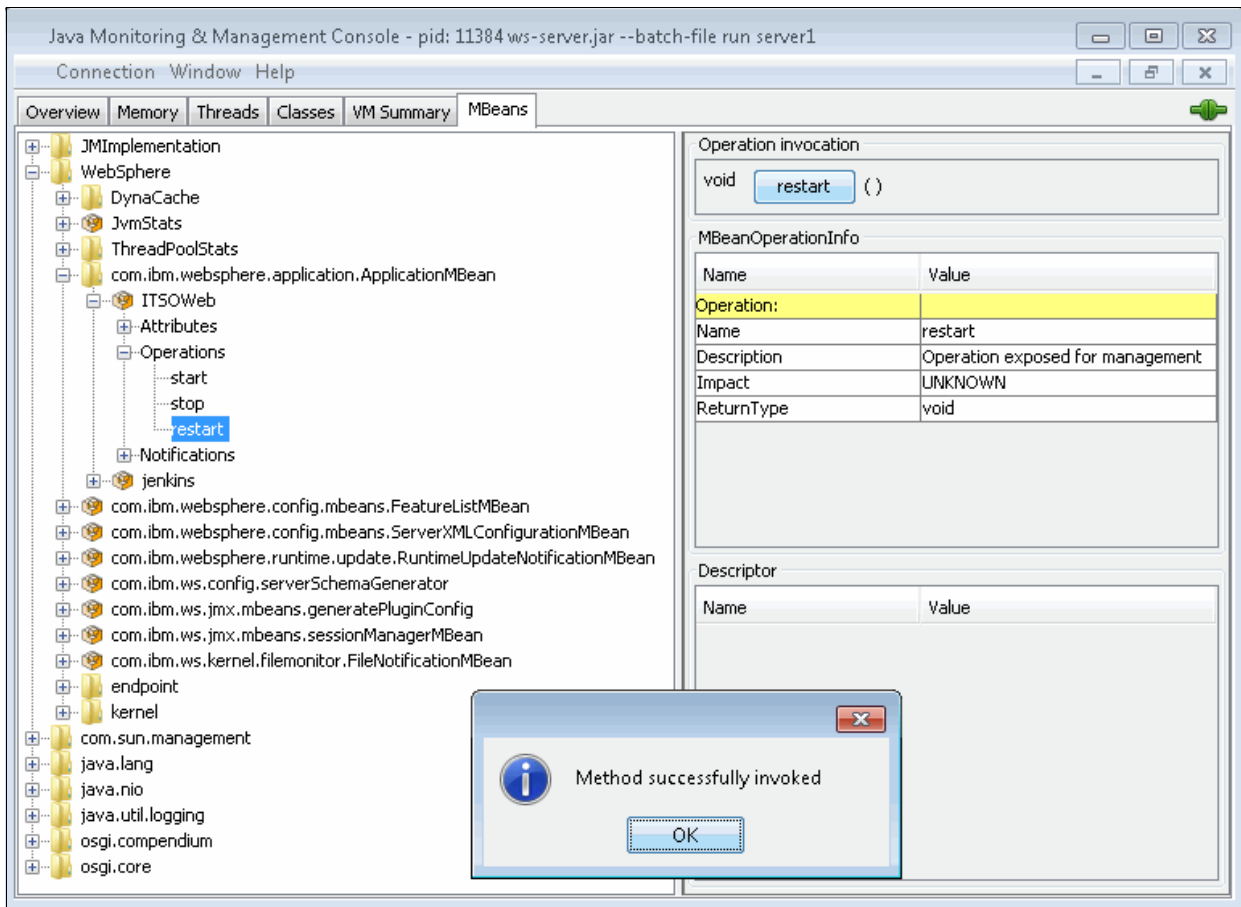


Figure 7-5 Issuing a restart request for an application using MBeans

If you enabled the monitoring infrastructure in Liberty, you can also access additional MXBeans such as **JvmStats**, as illustrated in Figure 7-6.

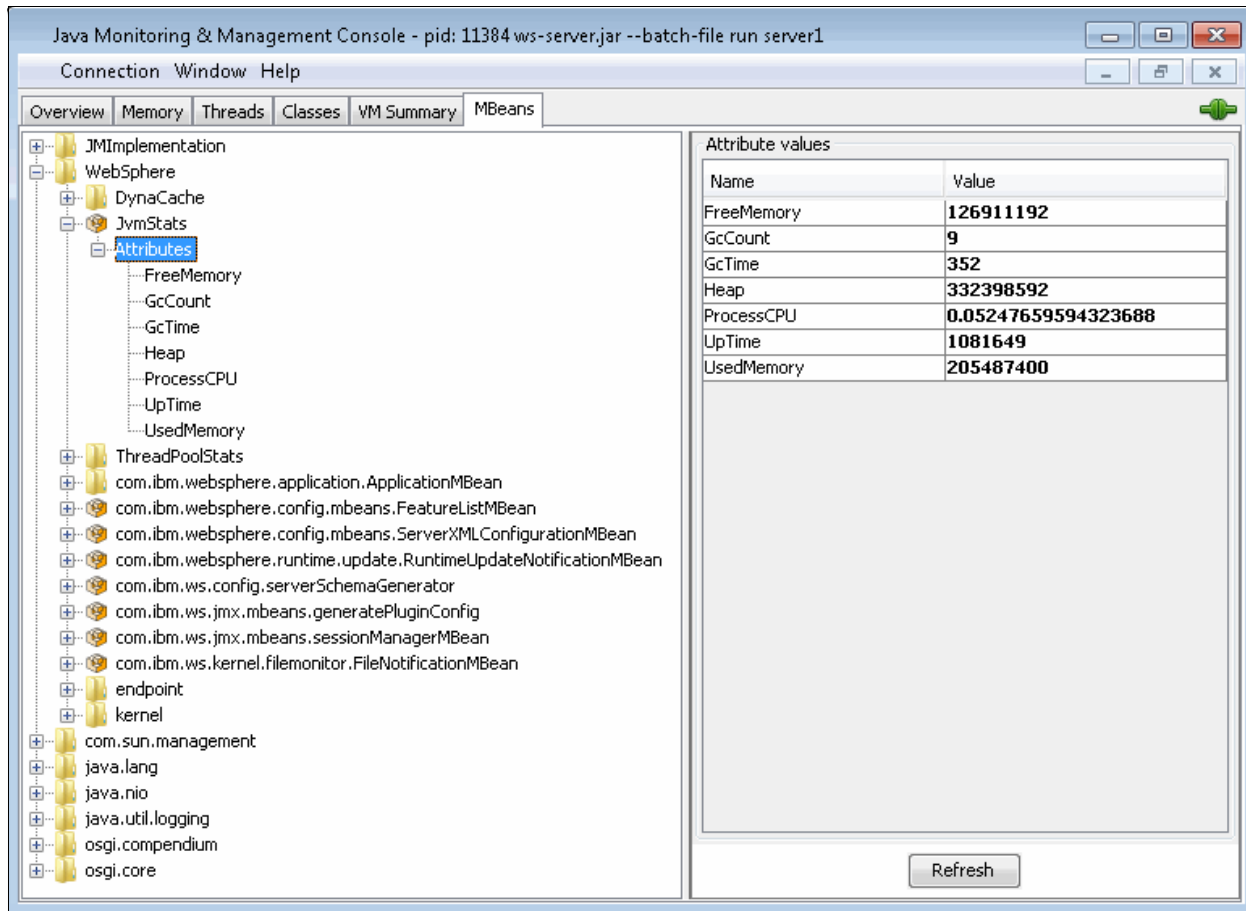


Figure 7-6 Viewing JVMStats using JConsole

7.4 OSGi Debug console

The Liberty profile server provides the `osgiConsole-1.0` feature that helps you administer the OSGi applications and features and also can be used for debugging. It can be used to display information about bundles, packages, and services that may be useful when developing your own features for product extensions.

To set the console, follow the steps in “OSGi console” on page 110.

After configuring the feature and restarting the server, you can access the OSGi console by issuing a **telnet** command to the console port as in Example 7-3.

Example 7-3

```
C:\>telnet localhost 5471
Connecting To localhost
osgi>
```

Type **help** to view a list of the options that are available in the OSGi console. For example, the **ss** command returns a summary status of all installed bundles.

7.5 Event logging feature

The new event logging feature helps you debug your Liberty profile applications by creating log entries for events such as JDBC, servlet, or HTTP requests. This feature enables logging events and their duration. You can control which event types are to be logged, the minimum duration to trigger the logging, the log mode, and the sample rate. This gives you the opportunity to debug slow running components of your applications and it can be a good starting point for tuning the applications and their runtime environment.

The eventLogging feature is not included in the runtime by default. To install the feature into your Liberty runtime, use the featureManager command as follows:

```
featureManager install eventLogging-1.0
```

To enable this feature, you need to add the eventLogging-1.0 feature to the featureManager section of your server.xml file. To configure this feature, you can set the following properties in the server.xml configuration file:

- ▶ **eventTypes:** This property lets you select which type of events are logged. You can log more than one type of event by including a comma-delimited list. By default, invocation of all types of events is logged.
- ▶ **logMode:** This property lets you select between entry, exit, or both entry and exit types of events to be logged. So, the available values for the logMode property can be entry, exit, or entryExit. To set this property, you need to include a line such as in Example 7-4 in the server.xml configuration file.

Example 7-4 The eventLogging logMode property

```
<eventLogging logMode="entryExit" />
```

- ▶ **sampleRate:** For example, to log the events from one of every five requests, you need to set a value of 5 for this property. The default value is 1, meaning that events from all requests are to be logged. To set this property, you need to include a line such as in Example 7-5.

Example 7-5 The eventLogging sampleRate property

```
<eventLogging sampleRate="2" />
```

- ▶ **minDuration:** This property lets you select the minimum duration of the events to be logged. The default value is 500 milliseconds. This means that by default, only exit events that are longer than 500 ms will be logged. To log all events, you need to set the value to 0. The value must contain a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). If no unit of time is specified, it defaults to milliseconds. So, for a value of 1 second, the value can be 1000, 1 s, or 1000 ms. To set this property, you need to include a line such as in Example 7-6.

Example 7-6 The eventLogging minDuration property

```
<eventLogging minDuration="1000" />
```

- ▶ **includeContextInfo:** This property lets you decide whether or not to include context information in the log entries. The default value is true, but you can turn it off by setting the false value as in Example 7-7 on page 219. You can configure this property in the server.xml configuration file of the Liberty profile server.

Example 7-7 The eventLogging includeContextInfo property

```
<eventLogging includeContextInfo="false" />
```

7.6 Slow request detection and hung request detection capabilities

The requestTiming-1.0 feature provides detailed information about any servlet requests that take longer than a configurable amount of time to run. It also automatically collects java cores from the server when requests are detected to be hung to facilitate problem determination.

To enable this feature, you have to add the requestTiming-1.0 short name to the list of features in the `server.xml` configuration file of your Liberty profile server. To configure this feature, you can set values for the following parameters:

- ▶ **slowRequestThreshold:** The value that you set for this parameter is the amount of time a request can run before it is considered slow. You can specify an integer value followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). Combined units of time are allowed, for example 1m1s, which means 61 seconds. The default value is 10 seconds. If you need to disable slow request timing, set the value to 0. Example 7-8 shows an example of configuring this parameter for 300 milliseconds. A request duration higher than the value you set for this parameter triggers the logger to capture the details about the request and the events that made up the request.

Example 7-8 The requestTiming slowRequestThreshold property

```
<requestTiming slowRequestThreshold="300ms" />
```

- ▶ **hungRequestThreshold:** The value that you can set for this parameter is the amount of time that a request can run before it is considered hung. You can specify an integer value followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). Combined units of time are allowed, for example 1m1s, which means 61 seconds. The default value is 10 minutes. If you need to disable hung request timing, set the value to 0. Example 7-9 shows an example of configuring this parameter for 1 minute. A request duration higher than the value you set for this parameter triggers a warning in the messages log and request details to be captured.

Example 7-9 The requestTiming hungRequestThreshold property

```
<requestTiming hungRequestThreshold="1m" />
```

- ▶ **sampleRate:** This parameter is the sampling rate for the request timing only. The default value is 1. To sample all requests, you need to set the value to 1. To sample one out of 10 requests, you need to set the value to 10 as shown in Example 7-10.

Example 7-10 The requestTiming sampleRate property

```
<requestTiming sampleRate="10" />
```

- ▶ **includeContextInfo:** This property lets you decide whether or not to include context information in the log entries. The default value is true, but you can turn it off by setting the false value as in Example 7-11 on page 220. You can configure this property in the `server.xml` configuration file of the Liberty profile server.

Example 7-11 The requestTiming includeContextInfo property

```
< requestTiming includeContextInfo="false" />
```

7.7 Timed Operations feature

You have the option to detect operations in the application server that are running more slowly than expected by enabling the Timed Operations feature. This feature will log warnings about those operations and can use the information that is listed to decide if anything is running slower or faster than you expect.

To enable the Timed Operations feature, add the following element declaration inside the `featureManager` element in your `server.xml` file:

```
<feature>timedOperations-1.0</feature>
```

The following properties are available for configuring this feature:

- ▶ **enableReport:** This boolean property lets you choose whether or not to generate a report to the logs detailing the 10 longest timed operations, grouped by type, and sorted within each group by expected duration. The default value is `true`.
- ▶ **maxNumberTimedOperations:** A warning is logged when the total number of timed operations reaches the integer value of this property. The default value is 10000.
- ▶ **reportFrequency:** This property determines the frequency, in hours, of generating reports to the logs detailing the 10 longest timed operations, grouped by type, and sorted within each group by average of actual duration. You have to specify a positive integer followed by the unit of time (hours) as in Example 7-12. The default frequency value is 24 hours.

Example 7-12 Timed operations feature configuration

```
<timedOperations maxNumberTimedOperations="10" reportFrequency="1h"/>
```



From development to production

The Liberty profile runtime fits the full range of production environments, from stand-alone servlet engines to massive Java EE deployments. By using the different Liberty server configurations to fit the different applications, you can eliminate the complexity of managing different applications servers. You can simplify your operations with a single set of management skills, tools, and processes. The Liberty profile can be used to develop and deploy on the same runtime, to any environment, at any scale. The applications can be moved easily between the traditional and the cloud environments, and the Liberty profile can be used to create an elastic hybrid cloud environment.

The chapter contains the following sections:

- ▶ Configuring a server for production use
- ▶ Using the package utility
- ▶ Moving an application to the full profile
- ▶ Using the Liberty profile on z/OS

8.1 Configuring a server for production use

The following advanced configuration settings should be considered when using a Liberty profile server in a production environment.

8.1.1 Turning off application monitoring

By default, application directories are monitored for any updates. Because applications are unlikely to change frequently in a production environment and because this monitoring can be resource-intensive, the preferred practice is to disable application monitoring.

Application monitoring can be disabled in the Liberty profile developer tools by editing the server configuration. In the design view, click **Application Monitoring**. In the Application Monitoring details window, clear the box next to Monitor application drop-in directory. Additionally, you might want to change the value of the drop-down menu for Application update trigger to disabled.

To disable application monitoring, by editing the `server.xml` file, add the following element:

```
<applicationMonitor updateTrigger="disabled" dropinsEnabled="false"/>
```

You can also change the value of the `updateTrigger` attribute to `mbean`. This allows you to update the application manually by using JMX.

8.1.2 Turning off configuration file monitoring

As with application monitoring, configuration file monitoring is enabled by default. To disable this function by editing the `server.xml` file, add the following element:

```
<config updateTrigger="disabled"/>
```

If you want configuration file monitoring to be enabled and avoid being resource-intensive, you can change the value of the `monitorInterval` attribute. The `monitorInterval` means the rate at which the server checks for configuration updates. To change the value of the `monitorInterval` attribute, add the following element:

```
<config monitorInterval="60s" updateTrigger="polled"/>
```

8.1.3 Generating a web server plug-in configuration

To use the Liberty profile server with an external web server, you must generate the web server plug-in configuration, whose properties can be specified in the server configuration.

To add plug-in configuration properties in the Liberty profile developer tools, load the `server.xml` file in the Design editor and click the server configuration. Then, click **Add** and select **Generate Plug-in**. You can specify properties such as ports in the Generate Plug-in details window.

To add plug-in configuration properties by editing the `server.xml` file, add a `pluginConfiguration` element to the file. The element supports the following attributes:

<code>webServerPort</code>	The port that the web server uses to listen for requests. The default value is 80.
----------------------------	--

<code>webServerSecurePort</code>	The port that the web server uses to listen for secure requests. The default value is 443.
<code>ipv6Preferred</code>	Whether to prefer ipv6. The default value is false.
<code>pluginInstallRoot</code>	The plug-in installation root.
<code>sslCertLabel</code>	SSL certificate label.
<code>sslKeyringLocation</code>	SSL key ring location.
<code>sslStashFileLocation</code>	SSL stash file location.

The actual plug-in configuration file must be generated either by using a utility in the Liberty profile developer tools, or by running an `mbean` command using JMX. In the tools, generate the file by right-clicking the server and selecting **Utilities** → **Generate Web Server Plug-in**.

To generate the plug-in configuration using `mbean`, use a JMX console, such as `jconsole`, to connect to the server. The `localConnector-1.0` feature must be enabled on the server. From the console, start the `defaultPluginConfig` generation MBean.

8.2 Using the package utility

Because a Liberty profile server is lightweight, it can be packaged easily with applications in a compressed file. This package can be stored, distributed to colleagues, and used to deploy the application to a different location or to another system. It can even be embedded in the product distribution. The deployment by replacing the package is the best practice for the Liberty profile.

The packaged Liberty profile is an archive file that contains one or more types of resources of the Liberty profile environment like the server configuration or the application. The types depend on the topology that is deployed. You can extract them manually or you can use an extraction tool to deploy the file to one or more systems. Alternatively, you can use the job manager or Liberty collective in the WebSphere Application Server Network Deployment product to deploy these images.

The package contains all of the binary files, server configuration, and applications necessary to distribute the server.

8.2.1 Packaging a Liberty profile server by using the WebSphere developer tools

Use the following procedure to package a Liberty profile server from the WebSphere developer tools:

1. Stop the server.
2. From the Servers view, right-click the server and select **Utilities** → **Package Server**.
3. Enter a file name for the archive.
4. Select whether you want to include all server content (including the server binary files) or only the server applications and configuration.
5. Click **Finish**.

8.2.2 Packaging a Liberty profile server from a command prompt

Use the following procedure to package a Liberty profile server from a command prompt:

1. Navigate to the Liberty profile server installation root directory.
2. Run the `bin/server package server_name` command.

The `package` command supports the following options:

- | | |
|------------------------|--|
| <code>--archive</code> | The name of the output file |
| <code>--include</code> | Include the entire server configuration by specifying <code>a11</code> . Include only the server applications and configuration by specifying <code>usr</code> . Include only resources that are required by the server by specifying <code>minify</code> , that can minimize the server by scanning runtime and selecting only the loaded features. |

8.2.3 Using the Job Manager to package and distribute Liberty profile servers

In WebSphere Application Server V8.5 Network Deployment, use the Job Manager to perform these functions:

- ▶ Package the Liberty profile runtime environments, configurations, and applications.
- ▶ Distribute and deploy a Liberty profile server and applications.
- ▶ Start embedded profile packages.

For more information about how to package the Liberty profile using the job manager, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multipatform.doc/ae/tagt_jobmgr_comp_server.html

8.2.4 Using the Liberty collective to distribute Liberty profile servers

In WebSphere Application Server V8.5 Network Deployment, you can configure the Liberty collective. In the Liberty collective, the `FileTransfer` and `FileService` MBeans can be used to perform file actions on any Liberty server in the Liberty collective. This includes both Liberty servers configured as collective controllers and those configured as collective members. You can distribute the package to the Liberty server by using these MBeans. Also, you can use the AdminCenter to distribute the package, which is already available and more user-friendly to use MBean than using manually.

For more information about the package distribution using the Liberty collective, see the following website:

http://www-01.ibm.com/support/knowledgecenter/api/content/n1/en-us/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/cwlp_collective_arch.html

8.3 Moving an application to the full profile

Applications that are developed on the Liberty profile server can also be redeployed to run on a WebSphere full profile server. The WebSphere full profile environment provides some features that are not available in the Liberty profile server. Applications that are developed in the full profile environment might not run on Liberty profile.

8.3.1 Programming model differences between full profile and Liberty profile

The WebSphere full profile server and the Liberty profile server differ in programming support for the following technologies:

- ▶ Java EE 6
- ▶ Java EE 7
- ▶ Enterprise Open Service Gateway initiative (OSGi)

For example, the Liberty profile already supports Java EE 7, but full profile does not yet. For OSGi applications, Liberty profile does not support Blueprint security.

For a more detailed comparison of technologies that are supported by the full profile and Liberty profile, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_prog_model_support.html

8.3.2 Configuration differences between full profile and Liberty profile

Although applications developed in the Liberty profile environment can be run on the full profile, there might be some configuration differences that must be addressed.

General concerns

In the Liberty profile, many properties that represent time values can be specified using units of time. For example, 1 hour can be specified as the value 1h. In full profile, time values are typically expressed as numeric values. When migrating application resources such as data sources or connection managers to full profile, you might need to modify property values to be specified as numbers.

Class loading

The Liberty profile server provides different methods of controlling class visibility than the full profile server. If your application requires you to configure advanced class loading behavior, you might need to reconfigure class loading in the full profile environment. You might also need to configure class loading in full profile if your application embeds classes that conflict with the full profile run time.

For more information about class loading in the full profile, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/trun_classload.html

Data sources

Some data source properties have different names:

- ▶ `ifxIFX_LOCK_MODE_WAIT` is `informixLockModeWait` in the full profile.
- ▶ `supplementalJDBCTrace` is `supplementalTrace` in the full profile.

Some data source properties have different default values:

- ▶ `beginTranForResultSetScrollingAPIs` is true by default in the Liberty profile.
- ▶ `beginTranForVendorAPIs` is true by default in the Liberty profile.
- ▶ `statementCacheSize` is 10 by default in the Liberty profile.

The Liberty profile allows `connectionSharing` to be configured to either `MatchOriginalRequest` or `MatchCurrentState`. By default, it is `MatchOriginalRequest`.

The Liberty profile allows `connectionSharing` to be configured in a finer grained manner, where individual connection properties can be matched based on the original connection request or current state of the connection. In the full profile, `connectionSharing` is a combination of bits representing which connection properties to match based on the current state of the connection.

In the full profile, a value of 0 means match all properties that are based on the original connection request, and a value of -1 means to match all properties that are based on the current state of the connection. The default value for the full profile is 1, which means that the isolation level is matched based on the current state of the connection and all other properties are matched based on the original connection request.

For more information about the differences for data sources of the full profile and the Liberty profile, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_ds_diff.html

Connection Manager

Some of the connection manager properties have different names in the full profile and Liberty profile. Table 8-1 shows the property naming differences.

Table 8-1 Properties with different names

Liberty profile property name	Full profile property name
<code>maxConnectionsPerThread</code>	<code>maxNumberOfMCsAllowableInThread</code>
<code>maxIdleTime</code>	<code>unusedTimeout</code>
<code>maxPoolSize</code>	<code>maxConnections</code>
<code>minPoolSize</code>	<code>minConnections</code>

There are also differences in the way timeout values for immediate or never (disabled) are specified. In Liberty profile, 0 represents an immediate timeout and -1 represents a disabled timeout. In full profile, -1 represents an immediate timeout and 0 represents a disabled timeout.

The value of the purge policy can also differ slightly between full profile and Liberty profile. There are three possible values in Liberty profile: `EntirePool`, `FailingConnectionOnly`, and `ValidateAllConnections`. The `EntirePool` option maps directly to the `EntirePool` option in full profile. `FailingConnectionOnly` maps to the `FailingConnectionOnly` option with the `defaultPretestOptimizationOverride` property set to `false` in the full profile. `ValidateAllConnections` corresponds to `FailingConnectionOnly` with the `defaultPretestOptimizationOverride` property set to `true` in full profile.

For more information about the differences for Connection Manager of the full profile and the Liberty profile, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_connpool_diff.html

Security

In the Liberty profile, you can configure user-to-role mappings and `runAs` users in the `application-bnd` element of the `server.xml` file. In the full profile, you can configure this only in `ibm-application-bnd.xml/xmi` or via administrative console.

Liberty profile has the following security limitations as compared to full profile:

- ▶ Not all public APIs and SPIs are supported. The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the IBM Knowledge Center, and is also available in a separate compressed file in one of the Javadoc subdirectories of the /dev directory of the server image.
- ▶ No horizontal propagation of the security attribute.
- ▶ No **SecurityAdmin** MBean support; therefore, methods such as clearing the authentication cache are not available.
- ▶ No Java 2 Connector (J2C) principal mapping modules support.
- ▶ No multiple security domain support.
- ▶ No security auditing subsystem that is part of the security infrastructure of the server.

For more information about the differences for the security of the full profile and the Liberty profile, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_sec_diff.html

Web Services Security

WS-Security in the Liberty profile is configured by using the WS-SecurityPolicy within the Web Service Definition Language (WSDL) file of a web service application. WS-Security, in the full profile, is configured and enabled by using a policy set.

WS-Security in Liberty profile supports the following WS-Security Policy namespaces:

- ▶ <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702>
- ▶ <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802>
- ▶ <http://schemas.xmlsoap.org/ws/2005/07/securitypolicy>

WS-Security, in full profile, supports the WS-Security policy namespace:

<http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512>

The Liberty profile supports more policy assertions than the full profile. To sign or encrypt a SupportingToken, such as a UsernameToken in the Liberty profile, you assert the token as SignedSupportingTokens, SignedEncryptedSupportingTokens, or EncryptedSupportingTokens. In the full profile, you must use an XPath expression to sign or encrypt a SupportingToken.

Some endorsing tokens are not supported in the full profile, including EndorsingSupportingTokens, SignedEndorsingSupportingTokens, EndorsingEncryptedSupportingTokens, and SignedEndorsingEncryptedSupportingTokens.

Both the Liberty profile and the full profile support the SymmetricBinding and AsymmetricBinding assertions. Only Liberty profile supports the TransportBinding assertion.

The IncludeToken assertion is enforced in the Liberty profile, but is ignored in the WS-Security runtime environment of the full profile.

The Liberty profile supports PasswordDigest and key derivation in the UsernameToken assertion. The full profile supports only PasswordText in a UsernameToken.

An unrecognized element in the Security header is rejected by the full profile. It is accepted by the Liberty profile.

For more information about the differences for the Web Service Security of the full profile and the Liberty profile, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_wssec_cxf_diff.html

Web applications

The Liberty profile server does not automatically expand web archive (WAR) files that are deployed to the server. The Java EE specification states that the `getRealPath()` method returns a null value if the content is being made available from a WAR file.

If your application relies on a result being returned by `getRealPath()`, you must deploy the application as an expanded web application, not as a WAR file. For example, you can manually extract the WAR file and copy the expanded application to the dropins directory.

JSP

Full profile supports a `useInMemory` configuration option to store only translated JSP files in memory. The `jsp-2.2` feature of Liberty profile does not support this option.

8.4 Using the Liberty profile on z/OS

WebSphere Application Server V8.5 provides features for administering a Liberty profile on a z/OS platform. You can use IBM MVS™ operator commands to create, start, stop, or modify the Liberty profile servers.

The IBM Installation Manager is used to install the Liberty profile on z/OS using a part of the `com.ibm.websphere.liberty.zOS.v85` package offering described in 2.2.3, “Installation on z/OS” on page 51.

Before creating a server or running the default server instance, you need to set the Java Runtime. The Liberty profile runtime searches for the Java command in the following order of properties: **JAVA_HOME**, **JRE_HOME**, and **PATH**.

You can set the environment variable by executing a command with the same syntax as in Example 8-1. This command exports the environment variable, but it is only valid in the command prompt shell that you are currently in.

Example 8-1 Exporting the PATH environment variable

```
export PATH=/usr/lpp/java/J7.0_64/bin/:/bin:/usr/sbin/:
```

Additionally, you can use the Liberty profile `server.env` configuration file to set up a specified Java Runtime. To configure the Java runtime using this file, add the **JAVA_HOME** parameter and the required value for your environment into the file as in Example 8-2 on page 229. If the `server.env` file does not exist, you must create it manually in the Liberty profile `etc` directory, which should be in the Liberty profile installation directory. You have to create the `etc` directory manually in case it does not exist. The `server.env` file in the `etc` directory is shared by all servers created from runtime. You can also place the `server.env` file in the home directory of a server to be used only by that server.

Example 8-2 JAVA_HOME variable in server.env

```
MTRES1 @ SC49:/u/mtres1/IBM/etc>cat ./server.env
JAVA_HOME=/usr/lpp/java/J7.0_64/
MTRES1 @ SC49:/u/mtres1/IBM/etc>
```

Note: The Liberty profile installation directory is often represented by the `${wlp.install.dir}` variable in configuration files.

After setting the Java Runtime, you can create and start a Liberty profile server as in Example 8-3. You can configure your server and add features using the `server.xml` file.

Example 8-3 Creating and starting a Liberty profile server

```
MTRES1 @ SC49:/u/mtres1/IBM/bin>./server create server1
Server server1 created.
MTRES1 @ SC49:/u/mtres1/IBM/bin>./server start server1
Starting server server1.
Server server1 started with process ID 67633642.
MTRES1 @ SC49:/u/mtres1/IBM/bin>
```

To list the version of the Liberty profile that you are using, run the `server version` command. To find information about the features of your Liberty profile, run the `productInfo featureInfo` command as in Example 8-4.

Example 8-4 Listing the version and features of a Liberty profile

```
MTRES1 @ SC49:/u/mtres1/IBM/bin>server version
WebSphere Application Server 8.5.5.6, WAS FOR Z/OS 8.5.5.6 (1.0.9.20150425-1300)
on IBM J9 VM, version pmz6470sr8fp10-20141219_01 (SR8 FP10) (en_US)
MTRES1 @ SC49:/u/mtres1/IBM/bin>./productInfo featureInfo
appSecurity-1.0 [1.1.0]
appSecurity-2.0 [1.0.0]
beanValidation-1.0 [1.0.0]
blueprint-1.0 [1.0.0]
cdi-1.0 [1.0.0]
clusterMember-1.0 [1.0.0]
collectiveController-1.0 [1.0.0]
collectiveMember-1.0 [1.0.0]
concurrent-1.0 [1.0.0]
distributedMap-1.0 [1.0.0]
ejbLite-3.1 [1.0.0]
jaxrs-1.1 [1.0.0]
jdbc-4.0 [1.0.0]
jndi-1.0 [1.0.0]
jpa-2.0 [1.0.0]
jsf-2.0 [1.0.0]
json-1.0 [1.0.0]
jsp-2.2 [1.0.0]
ldapRegistry-3.0 [1.0.0]
localConnector-1.0 [1.0.0]
managedBeans-1.0 [1.0.0]
monitor-1.0 [1.0.0]
oauth-2.0 [1.0.0]
osgi.jpa-1.0 [1.0.0]
osgiConsole-1.0 [1.0.0]
```

```
restConnector-1.0 [1.0.0]
serverStatus-1.0 [1.0.0]
servlet-3.0 [1.0.0]
sessionDatabase-1.0 [1.0.0]
ssl-1.0 [1.0.0]
timedOperations-1.0 [1.0.0]
wab-1.0 [1.0.0]
webCache-1.0 [1.0.0]
webProfile-6.0 [6.0.0]
zosSecurity-1.0 [1.0.0]
zosTransaction-1.0 [1.0.0]
zosWlm-1.0 [1.0.0]
MTRES1 @ SC49:/u/mtres1/IBM/bin>
```

Additional optional features provide enhanced integration with z/OS qualities of service:

- ▶ Classify inbound HTTP requests with Workload Manager (WLM)
- ▶ Use an IBM DB2® Type 2 driver with Resource Recovery Services (RRS) transaction management
- ▶ Authenticate users by using a System Authorization Facility (SAF) user registry
- ▶ Authorize users by using a SAF authorization provider

You can add more features to your Liberty profile installation by using the **featureManager** command. Be aware that some features are available starting with a specific version of the Liberty profile. For example, the z/OS Connect offering feature can be set up only if you are running Liberty profile version 8.5.5.2 with interim fix packs IFPI18279 and IFPI18379, or version 8.5.5.3. Make sure that you always run the most recent version of the Liberty profile.

To add features to your z/OS Liberty profile server, two steps need to be configured:

1. Install the new feature (if available for your version of z/OS Liberty profile server). This can be accomplished by using IBM Installation Manager or the **featureManager** command. For more information, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_inst_assets.html?cp=SSAW57_8.5.5%2F3-11-0-1-2&lang=en

2. Enable the new feature by editing the `server.xml` configuration file of your z/OS Liberty profile server and adding the feature short name as a feature xml tag.

For more information about administering the Liberty profile on z/OS, see the following IBM Knowledge Center website:

http://www-01.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.wlp.zseries.doc/ae/twlp_admin_zos.html?cp=SS7K4U_8.5.5%2F1-3-11-0-3-1-19

8.4.1 IBM z/OS Connect

The *IBM z/OS Connect* offering is a new Liberty profile feature that encapsulates calling z/OS target applications by using Representational State Transfer (REST) calls. Following are the benefits of z/OS Connect:

- ▶ Provides RESTful access to identify and invoke z/OS-based business assets in IBM CICS®, IBM IMS™, UNIX System Services, and classic batch environments, opening up these assets to cloud and mobile-based system of engagement environments.

- ▶ Functionality that is based on Liberty server technology, is lightweight and easily configurable, and provides z/OS differentiation with SAF security integration, z/OS WLM, and Resource Recovery Services (RRS) integration. WLM integration means different URIs can have varying levels of priority and performance criteria.
- ▶ As a feature in the Liberty profile server on z/OS, you can integrate z/OS Connect with z/OS standard system management, which can run as a started task and integrate with z/OS automated operations.
- ▶ Provides the ability to secure individual or groups of z/OS Connect services with SAF security in which only specific users or groups can have access to specific services.
- ▶ Provides the ability to uniformly track requests from cloud, mobile, and web environments by using z/OS System Management Facility (SMF) services. This tracking means that z/OS clients can use their existing processes for auditing and chargeback for requests from these environments.
- ▶ Enables the ability to do an automatic conversion of the request payload from JavaScript Object Notation (JSON) form on input to binary form consumable by z/OS applications such as Cobol, PL/I, and C. The reverse for the response from the z/OS application, converting the output from binary to JSON form is also true.

To enable the z/OS Connect feature, you have to install it first. Example 8-5 provides a way to use the **featureManager** command in order to enable the z/OS Connect feature using a local repository. The second thing that you need to do is to edit the `server.xml` file of your Liberty profile server and add the `zosConnect-1.0` feature.

Example 8-5 Installing the zosConnect-1.0 feature by using the featureManager command

```
MTRES1 @ SC49:/u/mtres1/IBM/bin>featureManager install zosConnect-1.0
--offlineOnly --location=/u/mtres1/kits/8556/repository/files/
```

Additional Features Terms & Conditions:

By clicking on the "I agree" button , you agree that the program code, samples, updates, fixes and related licensed materials such as keys and documentation ("Code") that you are about to download are subject to the terms of the license agreement that you accepted when you acquired the Program for which you are obtaining the Code. You further agree that you will install or use the Code solely as part of a Program for which you have a valid agreement or Proof of Entitlement. The terms "Program" and "Proof of Entitlement" have the same meaning as in the IBM International Program License Agreement ("IPLA"). The IPLA is available for reference at <http://www.ibm.com/software/sla/>

Select [1] I Agree, or [2] I do not Agree: 1

```
Step 1 of 4: Starting installation...
Step 2 of 4: Installing zosConnect-1.0...
Step 3 of 4: Cleaning up temporary files...
Step 4 of 4: Installation completed
CWWKF1017I: One or more features installed successfully: [zosConnect-1.0].
Start product validation...
Product validation completed successfully.
```

```
MTRES1 @ SC49:/u/mtres1/IBM/bin>
```

For more information about the IBM z/OS Connect feature, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_setup_zosconnect.html

8.4.2 WebSphere optimized local adapters for z/OS

The z/OS native applications can use the WebSphere optimized local adapters (WOLA) new Liberty profile feature for z/OS to make inbound calls to the application server enterprise beans and enable native programs to act as a server and accept requests from applications running in the WebSphere Liberty profile server.

To enable this feature, you need to install it as shown in Example 8-6 and enable it by editing the `server.xml` file of your server. You have to use the `zosLocalAdapters-1.0` short name of the feature.

Example 8-6 Installing the `zosLocalAdapters-1.0` feature using the `featureManager` command

```
MTRES1 @ SC49:/u/mtres1/IBM/bin>featureManager install zosLocalAdapters-1.0
--offlineOnly --location=/u/mtres1/kits/8556/repository/files/
```

Additional Features Terms & Conditions:

By clicking on the "I agree" button, you agree that the program code, samples, updates, fixes and related licensed materials such as keys and documentation ("Code") that you are about to download are subject to the terms of the license agreement that you accepted when you acquired the Program for which you are obtaining the Code. You further agree that you will install or use the Code solely as part of a Program for which you have a valid agreement or Proof of Entitlement. The terms "Program" and "Proof of Entitlement" have the same meaning as in the IBM International Program License Agreement ("IPLA"). The IPLA is available for reference at <http://www.ibm.com/software/sla/>

Select [1] I Agree, or [2] I do not Agree: 1

```
Step 1 of 5: Starting installation...
Step 2 of 5: Installing jca-1.6...
Step 3 of 5: Installing zosLocalAdapters-1.0...
Step 4 of 5: Cleaning up temporary files...
Step 5 of 5: Installation completed
CWWKF1017I: One or more features installed successfully: [jca-1.6,
zosLocalAdapters-1.0].
Start product validation...
Product validation completed successfully.
MTRES1 @ SC49:/u/mtres1/IBM/bin
```

In order to use this feature, edit the `server.xml` file of your Liberty profile server and create a connection factory entry to provide a JNDI name as shown in Example 8-7.

Example 8-7 `Server.xml` tags used for the `zosLocalAdapters-1.0` feature

```
<server>
  <featureManager>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
  </featureManager>
```

```
<!-- Local adapters connection factory definition -->
  <authData id="mtres" user="mtres1" password="{xor}MjY6LTM6Pg==" />
  <connectionFactory id="wolaCF" jndiName="eis/ola" containerAuthDataRef="mtres">
    <properties.ola/>
  </connectionFactory>
<!-- Provide WOLA server identity -->
  <zosLocalAdapters wolaGroup="LIB1" wolaName2="LIB2" wolaName3="LIB3" />
</server>
```

For more information about using the `zosLocalAdapters-1.0` feature, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_dat_enableconnector.html

8.4.3 Disabling z/OS operator console command handling

A new feature for the z/OS Liberty profile is the option to disable the z/OS operator console command handling. This needs to be done because listening and handling the z/OS operator console commands directly from your application or product extension feature is in conflict with the built-in Liberty for z/OS operator console command handler.

Follow these steps to disable the z/OS operator console command handling:

1. Create the `bootstrap.properties` file (if it does not exist in the server configuration directory).
2. Set `websphere.os.extension=zosNoConsoleExtensions-1.0` in the `bootstrap.properties` file.
3. Restart your z/OS Liberty profile server.

For more information, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_disable_zos_console.html



Developing and deploying custom features

Starting with version 8.5.5, IBM WebSphere Application Server Liberty Profile supports direct enhancement of the runtime with user-defined features deployed in product extensions. Creating your own features allows you to customize or extend the behavior of the runtime using system programming interfaces (SPIs) that applications do not have access to. User-defined features can also provide additional application programming interfaces (APIs) and services to deployed applications.

This chapter walks through the creation, packaging, and deployment of a custom feature in the following sections:

- ▶ Considerations for creating custom features
- ▶ Defining a custom feature

9.1 Considerations for creating custom features

User-defined features become an integrated part of the runtime, allowing you to do things you cannot do with an application. The ways that applications are effected are as follows:

- ▶ Features are installed and managed separately from business applications, keeping your runtime or product code separate from user code.
- ▶ Services that are provided by a user-defined feature can receive user specified configuration from the `server.xml` file, and that configuration can be viewed and modified in the WebSphere developer tools configuration editor.
- ▶ Classes in Open Service Gateway initiative (OSGi) bundles that are provisioned by a user-defined feature can use SPIs that are provided by the Liberty profile or by other extensions.
- ▶ OSGi bundles in a user-defined feature can augment or extend supported programming models. For example, a custom Trust Association Interceptor (TAI) can be provided by a user-defined feature.
- ▶ OSGi bundles in a user-defined feature can interact with OSGi services that are provided by the runtime and by other extensions. With correct metadata, OSGi services that are provided by user-defined features can also be shared with OSGi applications.

General instructions for writing features and receiving configuration properties can be found at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_admin_extend.html

Consider the following list before you choose to extend the Liberty profile runtime:

- ▶ Depending on your needs, features can run directly on the Liberty kernel with no dependence on application containers or other features. This approach provides a lightweight solution.
- ▶ Defining your own features can be powerful, but comes at the cost of portability. If portability between application servers is a primary concern, developing specification-compliant applications is more appropriate than creating product extensions and features.
- ▶ It is important to use only documented and supported externals to ensure that the Liberty profile can be serviced or upgraded with minimal disruption to your features and extensions. APIs, SPIs, and Javadoc are provided in the `${wlp.install.dir}/dev/` directory for use when developing and building your features.
- ▶ Features can use OSGi management of native library loading, allowing you to package libraries for multiple architectures in a single JAR file. OSGi loads the appropriate library at runtime.

Other details about supported externals are provided in the IBM Knowledge Center section *Liberty profile externals support* (considered as the authoritative source) at this website:

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_profile_externals.html

9.2 Defining a custom feature

A Liberty profile feature is a collection of resources, usually OSGi bundles, that provide classes and services to the Liberty profile runtime. A feature is defined using a text-based manifest file that follows the Subsystem Service metadata format in the OSGi Enterprise R5 specification.

9.2.1 Elements of a feature

To create a feature, either manually or using WebSphere developer tools, you essentially must create a manifest file, `example.mf`, in the `lib/features/` directory of a product extension. That manifest file must describe the feature with this type of information:

- ▶ What it is called
- ▶ How it is referenced
- ▶ Whether other features can use it

Also, the manifest enumerates the feature's content, including any bundles that should be provisioned if the feature is enabled.

For more information about feature manifests, including the full list of all supported manifest headers, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_feat_definition.html

Fixed-value manifest headers

There are a few headers in a feature manifest that must have specific, predefined values to allow the file to be recognized as a feature:

`IBM-Feature-Version: 2`

The version of the feature subsystem type. Required.

`Subsystem-Type: osgi.subsystem.feature`

The feature subsystem type. Required.

`Subsystem-ManifestVersion: 1`

The version of the OSGi Subsystem manifest format. Specifying this header is not required, but recommended.

Required manifest headers

The key elements of a feature definition (shown with a representative example value) are as follows:

`Subsystem-SymbolicName: com.example.product.feature-1.0`

The symbolic name of the feature, which follows the same style and syntax as the `Bundle-SymbolicName` header for bundles. Features can include other features, and when they do, they use the symbolic name to reference each other. The symbolic name of a feature should be sufficiently unique. It is safe to assume that two features with the same symbolic name are, in fact, the same feature. This header also supports the usage of attributes and directives, notably the `visibility` directive that is described in 9.2.2, "Visibility constraints for features, packages, and services" on page 239.

Subsystem-Content: com.example.product.bundle; version="[1,1.0.100]"

This header is the key to provisioning. It is a comma-separated list of resources that defines the feature's content. As with package declarations in a bundle manifest, each element of this list can specify additional directives, such as type, location, or version. These directives are described in more detail in 9.2.3, "Subsystem content: Writing a minify-compatible feature" on page 242.

Recommended headers for public features

Here are the recommended headers for public features:

IBM-ShortName: example-1.0

A representative short name that can be used with the extension name, in `server.xml`, to enable a feature. This is recommended to simplify the user configuration of public features.

Subsystem-Localization: OSGI-INF/110n/localization

The name and description of a feature can be globalized. Use this header to package properties files containing the translated values with your feature.

Subsystem-Name: %name

A short, localizable, and descriptive feature name. You can specify the value directly, or use a %propertyName format, which looks up the translated value in the file that is referenced by the Subsystem-Localization header.

Subsystem-Description: %description

A short, localizable, and feature description. You can specify the value directly or use a %propertyName format, which looks up the translated value in the file that is referenced by the Subsystem-Localization header.

Subsystem-Version: 1.0.0.qualifier

The version of the feature. See the OSGi core specification section 3.2.5 for the details about how this is defined.

IBM-AppliesTo: com.ibm.websphere.appserver

The Liberty version that this feature applies to. You can specify a list of comma-delimited items, such as `product_id`, `productVersion`, `productInstallType`, and `productEdition` as in Example 9-1. If more than one item is supplied, the value for `product_id` must be different for each one.

Example 9-1 IBM-AppliesTo manifest header

```
IBM-AppliesTo: com.ibm.websphere.appserver; productVersion=8.5.5.5;  
productInstallType=Archive; productEdition="BASE,DEVELOPERS,EXPRESS,ND"
```

Subsystem-License: <site-name>/<license-document>

The license type for this feature. If this header is specified and you do not specify the IBM-License-Agreement and IBM-License-Information headers, the value of this header is displayed to the user for acceptance when installing.

IBM-License-Agreement:lafiles/LA

The prefix of the location of the license agreement feature. It contains the path to the LA<*language*> files in the Liberty installation directory.

IBM-License-Information:lafiles/LI

The prefix of the location of the license information feature files. It contains the path to the LI<*language*> files in the Liberty installation directory.

Headers that influence provisioning behavior

The (not required) headers that influence provisioning behavior are:

IBM-API-Package: `javax.servlet; type="spec"`

A list of packages that follows the bundle manifest's `Export-Package` syntax and lists the packages that are provided by a feature that should be visible to applications. For further information, refer to 9.2.2, "Visibility constraints for features, packages, and services" on page 239.

IBM-API-Service: `com.example.service.FeatureServiceOne`

This header is used to indicate which services from the feature are visible to OSGi applications. The services that are listed in this header must be registered in the service registry by the bundles that comprise the feature. The service-name is the Java class or interface name of the service. Any specified attributes are interpreted as service properties that should be used when providing the service to OSGi applications. For further information, refer to 9.2.2, "Visibility constraints for features, packages, and services" on page 239.

IBM-SPI-Package: `com.example.spi.utils`

A list of packages that follows the bundle manifest's `Export-Package` syntax and lists packages that are provided by a feature that should be visible to other features in other extensions. For further information, refer to 9.2.2, "Visibility constraints for features, packages, and services" on page 239.

IBM-Provision-Capability: `<OSGi LDAP Filter>`

Some features provide a function that is only useful if other features are enabled. This header allows a feature to be auto-provisioned when its requirements, as specified in the feature definition using a standard OSGi LDAP filter, are satisfied. This header is described in more detail in 9.2.5, "Creating a Liberty feature manually" on page 249.

IBM-App-ForceRestart: `install, uninstall`

The presence of this header in a feature manifest causes applications to be restarted when this feature is either installed, uninstalled, or both.

9.2.2 Visibility constraints for features, packages, and services

The Liberty profile runtime imposes visibility constraints allowing both the core runtime and any installed extensions to evolve independently. By default, features and their associated bundles are considered internal details of the runtime or extension that provides them. The following list identifies what this default means:

- ▶ All features are private and cannot be specified in `server.xml` or included by features from other extensions.

- ▶ Exported packages are only visible to other bundles provided by the same extension.
- ▶ Services that are registered in the service registry are not available to OSGi applications. Usage of registered services by consumers from other extensions is limited by the ability of consuming code to resolve the service interface.

The `Subsystem-SymbolicName` header supports a `visibility` attribute that controls how the feature can be used and resolved by the runtime and product extensions. As shown in Figure 9-1, there are three possible values for the visibility attribute: `public`, `protected`, and `private` (the default).

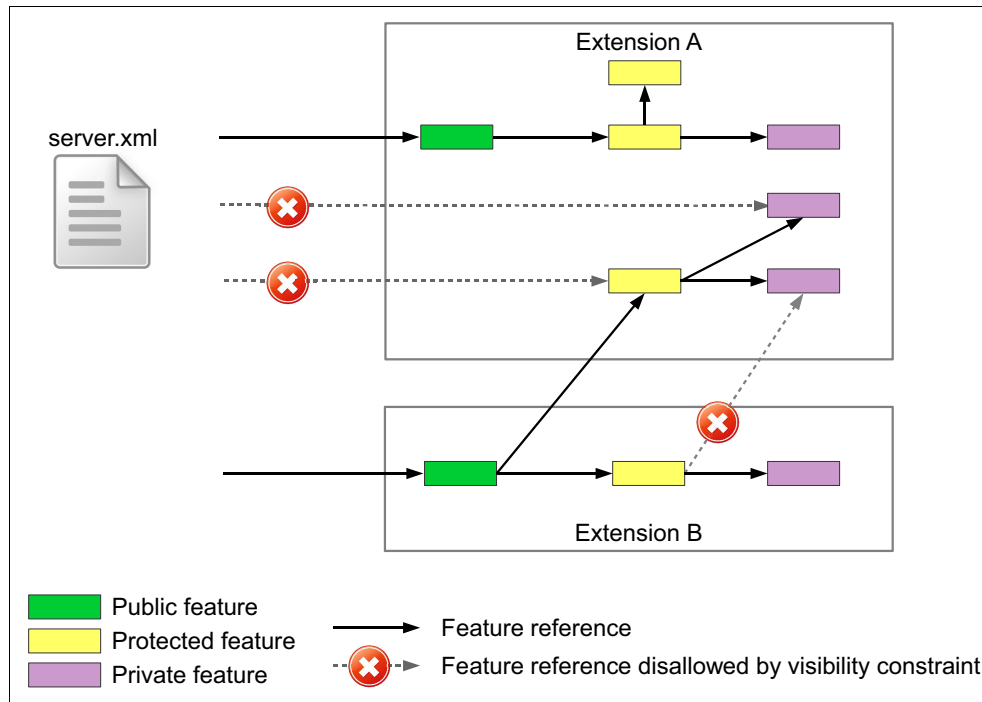


Figure 9-1 Visibility restrictions and features

Public features

Public features should be considered product externals or APIs. They can be used directly in `server.xml` files, are seen and used by the WebSphere developer tools, and appear in messages. Public features should have an `IBM-ShortName` header that is defined to simplify use. A list of public features that is provided by the Liberty runtime can be found in the following IBM support website:

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_feat.html

A public feature might be superseded by a newer version if necessary, such that the older version that is used by existing server configurations continue to function. For more information about superseded features, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/cwlp_feat_mgmt.html

Example 9-2 on page 241 shows a snippet from a public feature manifest. Observe the presence of the `IBM-ShortName` header, used in combination with the extension prefix in `server.xml`, and the public visibility constraint on the `Subsystem-SymbolicName` header.

Example 9-2 Declaration of a public feature, with an IBM-ShortName header

```
IBM-ShortName: example-1.0
Subsystem-SymbolicName: com.ibm.example.feature.public-1.0;
    visibility:=public
```

Protected features

Protected features are also product externals, but should consider SPI, as the target consumer for a protected feature is another feature. Although protected features are not directly referenced in a configuration, they are referenced by other extensions and should be treated with an equal amount of care when changes are made to avoid breaking consumers.

The Liberty profile defines several protected features, including features for the application manager and the class loading service, for use by product extensions. A list of protected features provided by the Liberty runtime area are available in the IBM support website:

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/rwlp_protected_features.html

Example 9-3 shows a snippet from a protected feature manifest.

Example 9-3 Declaration of a protected feature

```
Subsystem-SymbolicName: com.ibm.example.feature.protected-1.0;
    visibility:=protected
```

Private features

Private features are the default. Private features are usable only within a product extension. Implementation details should be packaged using private features. Auto features, described in 9.2.5, “Creating a Liberty feature manually” on page 249, are almost always private features. They rely on the provisioning of other public or protected features to automatically enable additional capabilities.

Example 9-4 shows a snippet from a private feature manifest.

Example 9-4 Declaration of a private feature, which omits the visibility attribute

```
Subsystem-SymbolicName: com.ibm.example.feature.private-1.0
```

Declaring API and SPI packages that are provided by a feature

By default, the individual packages that are exported by the bundles provisioned by a feature are private to the feature’s extension. This means that the individual packages that the bundles use to comprise your features are treated as internal implementation details for your product extension. Packages that should be shared with applications (as API) or with bundles from features in other extensions (as SPI) must be explicitly declared using manifest headers that follow the bundle’s `Export-Package` syntax.

The `IBM-SPI-Package` header is used to list packages that are provided by a feature that should be treated as SPI. The `IBM-API-Package` header is used to list packages that are provided by a feature that should be treated as API. Both headers support an optional type attribute that identifies the type of package that is being shared. The type attribute uses the following values:

<code>api</code>	This value applies only to the <code>IBM-API-Package</code> header and is the default type for API package declarations. It is used to mark user-defined or product-defined API.
------------------	--

ibm-api	This value applies only to the IBM-API-Package header and is used for IBM-provided API packages.
internal	This value applies only to the IBM-API-Package header and is used to indicate packages that are not API packages. However, they must be available to application classloaders in order for them to function correctly. For example, because of the use of reflection, bytecode enhancement or weaving, and resource injection.
spec	Used for API or SPI provided by a standards body such as, javax.servlet or org.osgi.framework.
third-party	Used for API or SPI to show that it is visible. However, these items are not provided or controlled by IBM. This situation usually applies to open source packages. Use of third-party packages carries additional risk for version-to-version migration, as these packages might change over time without consideration for compatibility with earlier versions.

Note: When enumerating API and SPI packages, the package version is not specified. The package versions should be maintained only by the bundles that provides those packages.

You can see all feature manifest files of your Liberty profile in the /lib/features directory. For example, the values of the restConnector-1.0 feature are:

IBM-API-Package: com.ibm.websphere.jmx.connector.rest;
type="ibm-api",com.ibm.ws.jmx.connector.client.rest;
type="ibm-api",com.ibm.websphere.filetransfer; type="ibm-api"

IBM-SPI-Package: com.ibm.wsspi.collective.plugins; type="ibm-spi",
com.ibm.wsspi.collective.plugins.helpers; type="ibm-spi"

9.2.3 Subsystem content: Writing a minify-compatible feature

The package utility, which is mentioned in 8.2, “Using the package utility” on page 223, can produce a *minified* server image. When you package a server with the `--include=minify` option, the package utility creates an archive that contains resources that only the server configuration requires. The list of required resources is built using the Subsystem-Content header that is defined for each enabled feature. When you create your own feature, it is important to verify the value of this header to ensure that a minified server that uses your feature has all of the resources that it needs.

The Subsystem-Content header follows the syntax shown in Example 9-5. As with an Export-Package bundle header, it is a comma-separated list of values, with optional semi-colon separated parameters, which can be attributes (=) or directives (:=). Each item in the list identifies an individual resource using a unique name.

Example 9-5 Syntax for Subsystem-Content header (see OSGi core spec section 1.3.2)

```
Subsystem-Content ::= resource ( ',' resource )*
resource ::= unique-name ( ';' parameter )*
unique-name ::= identifier ( '.' identifier )*
parameter ::= directive | attribute
directive ::= extended ':=' argument
attribute ::= extended '=' argument
```

```

argument ::= extended | quoted-string
extended ::= ( alphanum | '_' | '-' | '.' )+
quoted-string ::= '"' ( ~["\#x0D#x0A#x00] | '\\" | '\\\' )* '"'

```

The name that is used for a resource is influenced by its type, which is specified as an optional attribute. By default, a resource is assumed to be an OSGi bundle (`type="osgi.bundle"`). The bundle's symbolic name is used when specifying a bundle resource. Likewise, a feature's symbolic name is used, along with `type="osgi.subsystem.feature"`, to include another feature as a resource. For other supported types (`type="file"` and `type="jar"`) a unique string should be created for the resource using the syntax and character set that is allowed for symbolic or package names.

Resources can also specify a location directive to indicate where the artifact for that resource can or should be found. The specified location is also dependent on the type. The directive is not required for bundles or features, as the default (`lib/` and `lib/features/`, respectively) is sufficient. For bundles and JAR files, the location directive can point to a list of directory locations, which constitute a search path (such as, look in `dev/` first, then in `lib/`). The core runtime uses this list mechanism to provide API and SPI JAR files in the `dev/` directory that are then serviced by the `lib/` directory. For JAR and plain files, the location can (and in the case of files, must) contain a single path pointing directly to the backing resource. Paths are specified relative to the installation root of the extension containing the feature. When referencing public or protected features from another extension, the location field should not be specified. Rely on the symbolic name of the feature you are referencing and let the runtime figure out where to load it from.

In addition to type, the following list notes other supported directives:

<code>version</code>	The version attribute applies to bundles and specifies the versions to be matched when finding a bundle to satisfy the feature. Only bundles in this range are selected. A typical example of the version range is <code>(1,1.0.100)</code> .
<code>start-phase</code>	Start-phase is a directive that indicates when (relative to the rest of the runtime) a bundle that is used by a feature should be started. The start-phase directive can take one of the following values: <code>SERVICE</code> , <code>CONTAINER</code> , or <code>APPLICATION</code> . Bundles can also be defined to start before or just after these phases by adding <code>_LATE</code> to be later or <code>_EARLY</code> to be earlier than the key phase. So if you want to run immediately after the container phase, use <code>CONTAINER_LATE</code> , and if you want to run before the <code>APPLICATION</code> phase, use <code>APPLICATION_EARLY</code> .

The next sections provide examples of these directives in use.

9.2.4 Using the tools to create a custom feature

In this section, we describe how to build a simple feature containing a single bundle that prints "Hello World!" when it is started, and "Goodbye World!" when it is stopped. The focus here is on creating a feature and the supporting bundle resource, publishing the feature to a product extension, and finally using the feature as the catalyst to cause our bundle to be loaded and started by the OSGi runtime.

Creating an OSGi bundle project

Begin by creating a project for this bundle by completing the following steps:

1. Click **File** → **New** → **OSGi Bundle Project**.

2. Enter a project name. This example uses the project name `ITSO.OSGiExample.SimpleBundle`.
3. For the Target Runtime, select **WebSphere Application Server V8.5 Liberty Profile**.
4. Clear the **Add bundle to application** check box. If you leave this selected, a new OSGi application project is created.
5. Leave all other values at the default settings and click **Next** → **Next** accepting all defaults.
6. This example uses a simple BundleActivator to show that the bundle is loaded by the runtime. Select the **Generate an activator** check box.
7. Exit the wizard by clicking **Finish**. The contents of the new bundle project should resemble what is shown in Figure 9-2.

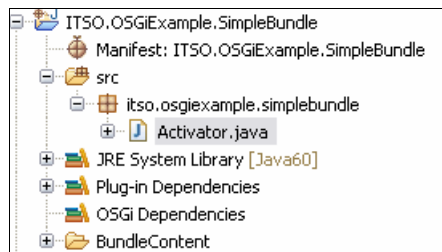


Figure 9-2 OSGi bundle project with generated Activator

8. Update the contents of the Activator to roughly match the example contents of Figure 9-3.

```

package itso.osgiexample.simplebundle;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        System.out.println("Hello World!");
    }

    public void stop(BundleContext context) throws Exception {
        System.out.println("Goodbye World!");
    }
}

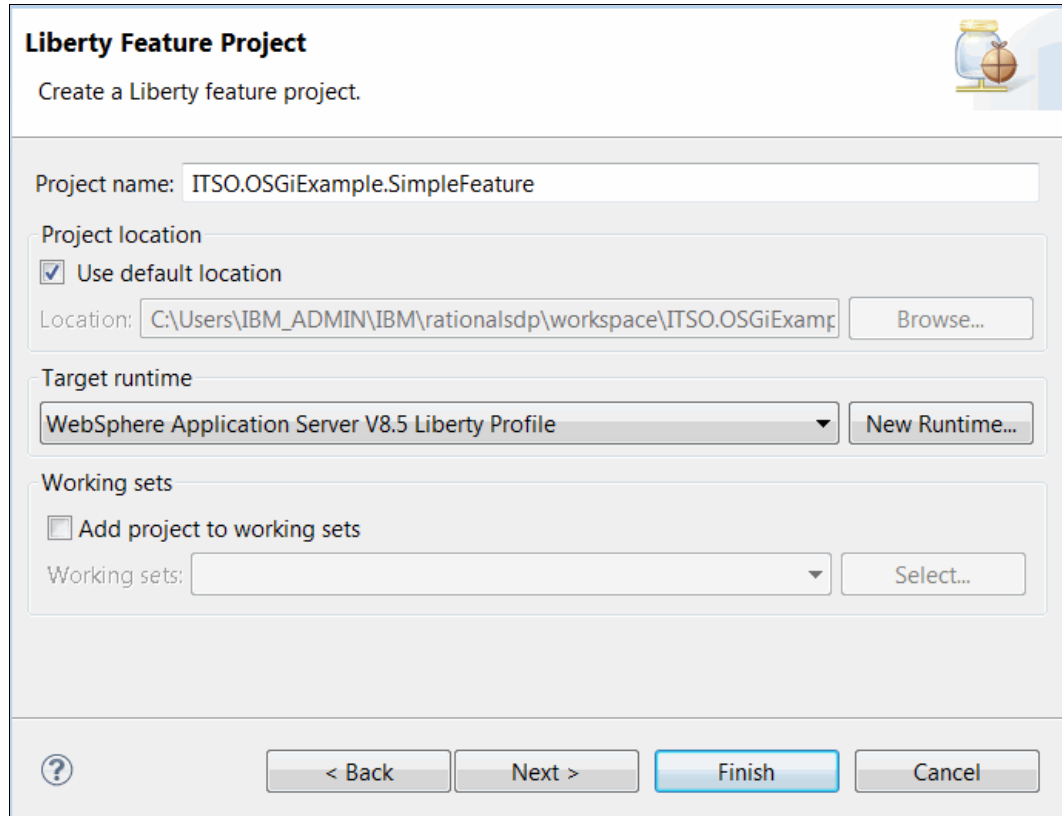
```

Figure 9-3 Simple bundle activator

Creating a Liberty feature project

Begin by creating a project for the new feature by completing the following steps:

1. Click **File** → **New** → **Other** → **OSGi Liberty Feature Project** and then click **Next** to open the Liberty Feature Project wizard, as shown in Figure 9-4 on page 245.



The screenshot shows the 'Liberty Feature Project' wizard dialog box. The title bar reads 'Liberty Feature Project' with a small icon of a globe on a stand. Below the title, it says 'Create a Liberty feature project.' The dialog is divided into several sections:

- Project name:** A text field containing 'ITSO.OSGiExample.SimpleFeature'.
- Project location:** A section with a checked checkbox 'Use default location'. Below it, a 'Location:' text field contains 'C:\Users\IBM_ADMIN\IBM\rational\workspace\ITSO.OSGiExam' and a 'Browse...' button.
- Target runtime:** A dropdown menu showing 'WebSphere Application Server V8.5 Liberty Profile' and a 'New Runtime...' button.
- Working sets:** A section with an unchecked checkbox 'Add project to working sets'. Below it, a 'Working sets:' text field is empty and a 'Select...' button is present.

At the bottom of the dialog, there is a help icon (question mark) on the left and four buttons: '< Back', 'Next >', 'Finish' (highlighted in blue), and 'Cancel'.

Figure 9-4 The Liberty Feature Project wizard

2. Enter a project name. This example uses the project name ITSO.OSGiExample.SimpleFeature.
3. For the Target runtime, select **WebSphere Application Server V8.5 Liberty Profile** and click **Next**.

4. Select the check box next to **itso.osgiexample.simplebundle** to include our example bundle in our new feature as shown in Figure 9-5.

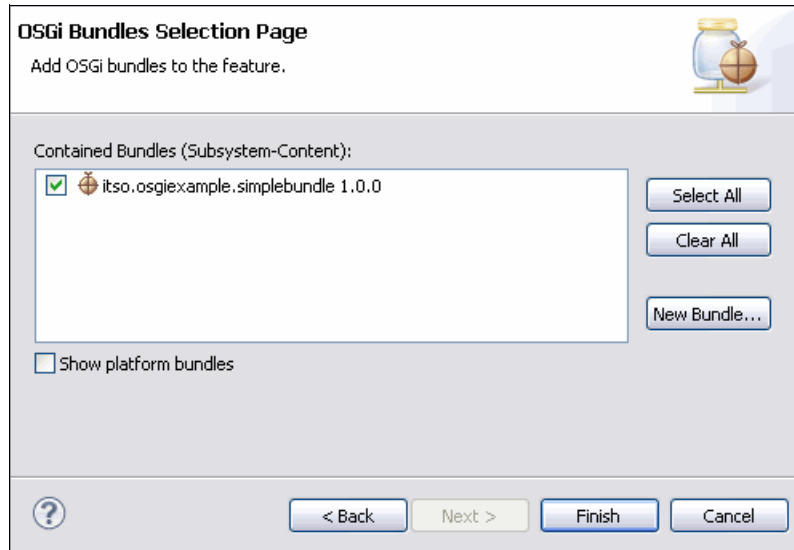


Figure 9-5 Adding a bundle to a feature

5. Exit the wizard by clicking **Finish**. The contents of the new feature project should resemble what is shown in Figure 9-6.

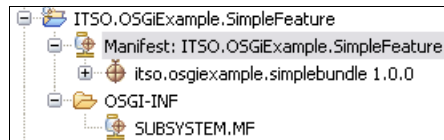


Figure 9-6 Liberty feature project

- Double-click the **Manifest element** in the feature project to open the manifest editor. Update the values in the manifest (Figure 9-7) and save your changes. Specifically, update the feature's symbolic name and the feature name to include a version representation. This is counter to how versions usually work. In this case, consider the version string in the feature name to be a statement of API/SPI compatibility, especially for public features.

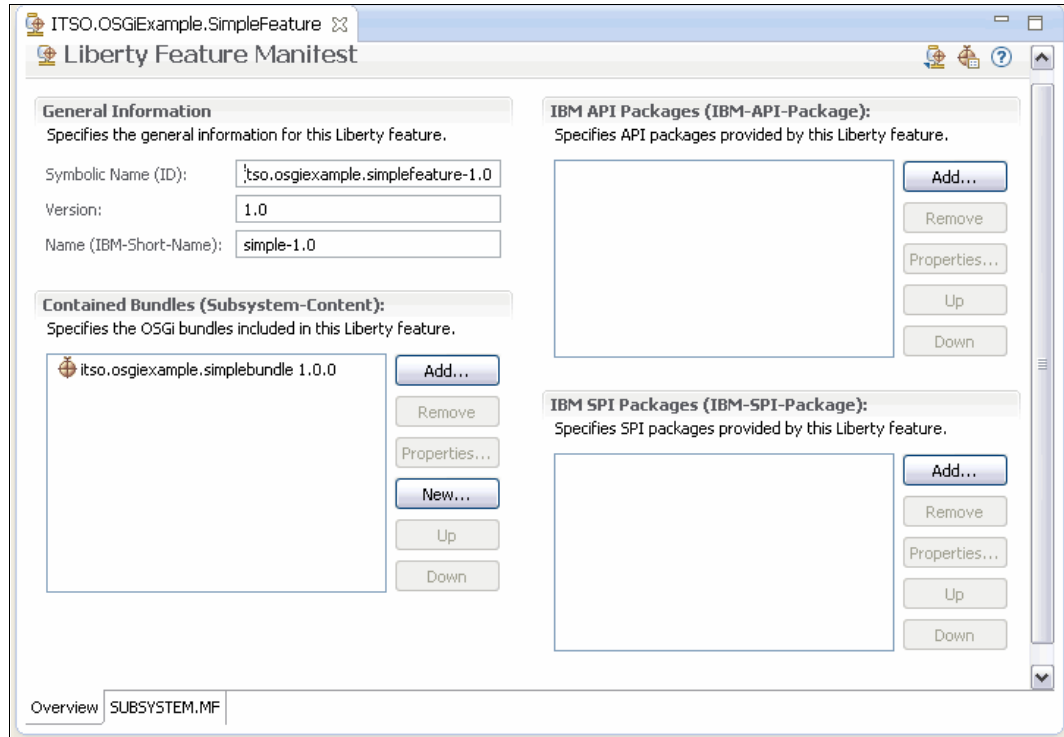


Figure 9-7 The feature manifest editor

- To support making changes to the bundle version (to apply service), select the bundle from the **Subsystem-Content** section of the editor, click **Properties**, and edit the **Minimum Version** and **Maximum Version** fields (Figure 9-8 on page 247). Click **OK**.

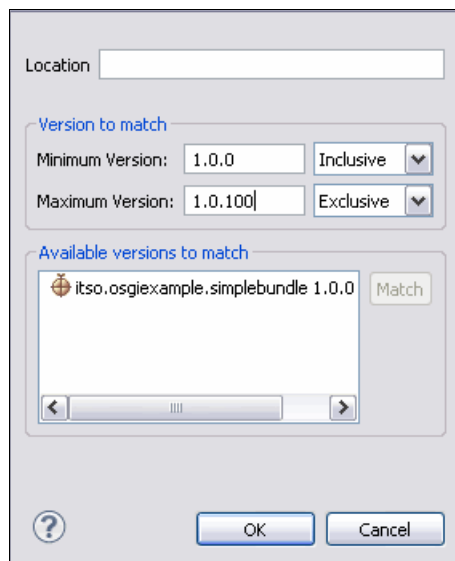


Figure 9-8 Editing the properties of a bundle that are listed in a feature's Subsystem-Content

- Click the **SUBSYSTEM.MF** tab at the bottom of the manifest editor to view the raw source of the manifest, resembling what is shown in Figure 9-9. The specified Subsystem-Content is minimal, as the feature includes only a bundle resource (which is the default type) in a default location with the specified required version range.

```
Subsystem-ManifestVersion: 1.0
IBM-Feature-Version: 2
IBM-ShortName: simple-1.0
Subsystem-SymbolicName: itso.osgiexample.simplefeature-1.0;visibility:=public
Subsystem-Version: 1.0
Subsystem-Type: osgi.subsystem.feature
Subsystem-Content: itso.osgiexample.simplebundle;version="[1.0.0,1.0.100]"
Manifest-Version: 1.0
```

Figure 9-9 Raw contents of the feature manifest

Installing the feature

The WebSphere developer tools make installing the feature painless. Right-click the feature project, select **Install Feature** from the menu, select **WebSphere Application Server V8.5 Liberty Profile** for the target run time, and click **Finish**. The WebSphere developer tools adds the feature and its contents to the default product extension of the target run time (Figure 9-10). You need to restart the server in case it is running.

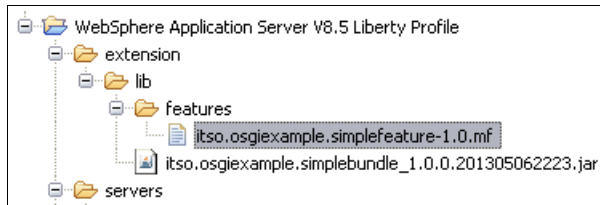


Figure 9-10 The example bundle and feature that is installed into the default product extension

Enabling the feature in a test server

Use the server configuration editor to edit the configuration of a test server by clicking **Add** next to the **Feature Manager** list and selecting the feature that we created from the list. Because it is a feature in the default product extension, it appears in the list with a `usr:` prefix. This example shows `usr:simple-1.0` (Figure 9-11). Click **OK** to finish, and save the updated server configuration.

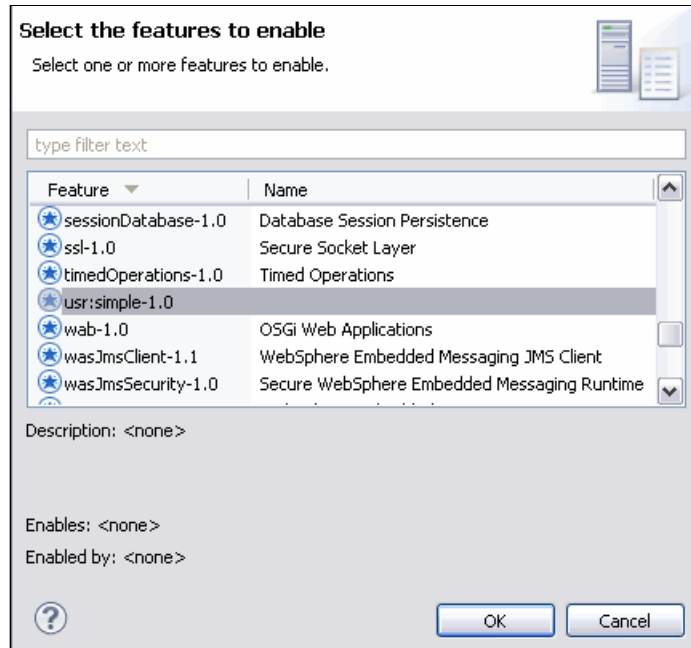


Figure 9-11 The `simple-1.0` feature appears in the feature list with the `usr:` prefix

If you add this feature to a running server, you see something similar to the output shown in Example 9-6. “Hello World!” is the output from the simple activator that we provided for the feature bundle.

Example 9-6 Console output after enabling the `usr:simple-1.0` feature.

```
[AUDIT ] CWWKF0011I: The server simpleServer is ready to run a smarter planet.  
[AUDIT ] CWWKG0016I: Starting server configuration update.  
Hello World!  
[AUDIT ] CWWKF0012I: The server installed the following features:  
[usr:simple-1.0].  
[AUDIT ] CWWKF0008I: Feature update completed in 0.220 seconds.  
[AUDIT ] CWWKG0017I: The server configuration was successfully updated in 0.236  
seconds.
```

9.2.5 Creating a Liberty feature manually

You can always create and deploy a Liberty feature manually, without the need of WebSphere Tools.

The two things that you need in order to obtain a Liberty feature are the OSGi bundle and the corresponding feature manifest file. The OSGi bundle has to be copied in the `${wlp.user.dir}/extension/lib` directory and the manifest file has to be copied into the `${wlp.user.dir}/extension/lib/features` directory. By doing these two things, you are able

to finally install the new feature by adding its short name into the `server.xml` configuration file of the Liberty profile server.

To create a Liberty feature manually, follow the steps below:

1. Develop the OSGi bundle and the bundle manifest. It is required that the OSGi bundle manifest file contains the `Bundle-SymbolicName` header to specify a unique name for the bundle. Use the `Export-Package` header to specify the exported packages as in Example 9-7.

Example 9-7 Manifest file of an OSGi bundle

```
Bundle-SymbolicName: com.morleyc.example
Bundle-Version: 1.0.7
Bundle-ManifestVersion: 3
Export-Package: com.morleyc.example.testp1; version="1.0.3",
                com.morleyc.example.testp2; version="1.0.1",
                com.morleyc.example.testp3; version="1.0.7"
```

2. Package the OSGi bundle Java classes and the manifest file by using the `jar` command as in Example 9-8.

Example 9-8 Packaging the Java classes and the manifest file by using the jar command

```
jar cfm example.jar MANIFEST.Mf *.class
```

3. Create the manifest file of the new Liberty profile custom feature by adding the required and the recommended headers as explained in 9.2.1, “Elements of a feature” on page 237. Example 9-9 shows such a manifest file named `new-feature-1.0.mf`.

Example 9-9 The contents of the custom Liberty profile feature new-feature-1.0.mf file

```
Subsystem-ManifestVersion: 1.0
Subsystem-SymbolicName: com.morleyc.example.new-feature-1.0; visibility:=public
Subsystem-Version: 1.0.0.qualifier
Subsystem-Type: osgi.subsystem.feature
Subsystem-Content: example; version="[1,1.0.100)"
IBM-Feature-Version: 7
IBM-API-Package: com.morleyc.example.testp1; type="api",
                com.morleyc.example.testp2; type="api",
                com.morleyc.example.testp3; type="api"
IBM-SPI-Package: com.new-feature.myservice.spi;
IBM-ShortName: new-feature-1.0
```

4. Copy the bundle of the new Liberty profile feature into the `${wlp.user.dir}/extension/lib` directory.
5. Copy the new Liberty profile feature manifest file into the `${wlp.user.dir}/extension/lib/features` directory.
6. If defined by the `Subsystem-Name`, `Subsystem-Description`, and `Subsystem-Localization` headers, copy the localization files into the `${wlp.user.dir}/extension/lib/features/110n` directory.
7. Add the new feature to the Liberty profile server `server.xml` configuration file as in Example 9-10.

Example 9-10 Adding the new feature in the server.xml Liberty profile configuration file

```
<featureManager>
  <feature>usr:new-feature-1.0</feature>
```

</featureManager>

9.2.6 Automatic provisioning: Creating an auto-feature

An automatic feature is not configured by the user in the server XML. It is automatically loaded by the feature manager if the features specified in its `IBM-Provision-Capability` manifest header are configured by the user. This is useful when two features require additional Java classes to work together. The additional classes are put into the automatic feature. When only one of those features is configured, the extra Java classes are not loaded. In this way, automatic features can be used to keep features as small as possible without requiring the user to understand any relationships between them.

For example, one feature might provide an application container and another might provide a generic security service. In order for them to work together, security collaborators specific to that container are required. Those collaborators are not useful in the absence of either the container or the security service. In this example, the security collaborator classes are packaged into a third, automatic feature, which contains the entry in its manifest file shown in Example 9-11.

Example 9-11 Entry for automatic feature

```
IBM-Provision-Capability:  
osgi.identity;  
filter:="(&(type=osgi.subsystem.feature)(osgi.identity=com.ibm.itso.container))",  
osgi.identity;  
filter:="(&(type=osgi.subsystem.feature)(osgi.identity=com.ibm.itso.security))"
```

The syntax for the `IBM-Provision-Capability` header is an LDAP query. The feature names that are specified in the query must be the `Subsystem-SymbolicName` values, not the `IBM-ShortName` values.

The bundles in the automatic feature can have mandatory dependencies on the bundles in both of the other features because the feature is provisioned only when both the container and security features are present. This is another way that automatic features helps minimize the mandatory dependencies between configured features.

9.2.7 Packaging native code in your bundles

OSGi provides help for loading native libraries. You can package libraries for all supported architectures in your bundle and list them on the `Bundle-NativeCode` manifest header. Code a single `System.loadLibrary` call in your Java code. OSGi examines the manifest header, determines the correct library for the architecture being used, and loads that library. Example 9-12 shows an example definition of the native library name for Windows and OSX.

Example 9-12 Definition of the native library name for Windows and OSX

```
Bundle-NativeCode:  
lib/win32/x86/name.dll; osname=win32; processor=x86,  
lib/macosx/libname.dylib; osname=macosx; processor=x86; processor=x86_64
```

Java has some limitations when loading native libraries that also apply in an OSGi framework. For example, a native library can be loaded only by one classloader in a JVM process, so the same native library cannot be used by more than one OSGi bundle.

9.2.8 Packaging features for delivery

Features can be packaged individually for an application to an existing Liberty profile installation or can be included in a packaged server if a self-contained deployment package is required.

OSGi subsystem archive

A subsystem archive is the OSGi-specified package for an individual feature. It contains the feature (subsystem) manifest and the files that are described by the Subsystem-Content manifest header. It is an archive file with an `.esa` suffix.

A subsystem archive is the output of WebSphere developer tools when you select Export for a Liberty feature project and can be applied to a Liberty profile installation by running the `featureManager install` command in the `bin` directory.

Packaged server

If you want to create a complete deployment solution, you can generate an archive by running the `server package` command. This outputs a compressed file with the following contents:

- ▶ The Liberty runtime
- ▶ Any features in the user product extension directory (`wlp/usr/extension`)
- ▶ Your server configuration (for the server that is specified on the `server package` command)
- ▶ Any applications and resources that are contained within the server

Using the `minify` option generates a smaller package containing only the features that are used by the server as in Example 9-13. This includes features that are configured in the `server.xml` file, features that are included by configured features, and features that are resolved as automatic features from the configured features. For example, a minified server package for servlet support is only about 17 MB.

Example 9-13 Using the `server package` command with and without the `minify` option

```
C:\IBM\WebSphere\Liberty855_OSGI\bin>server package server --include=minify
Packaging server server.
Querying server server for content.
Launching server (WebSphere Application Server 8.5.5.5/wlp-1.0.8.c150520150305-2202) on Java HotSpot(TM) 64-Bit Server
VM, version 1.7.0_79-b15 (en_US)
[AUDIT ] CWWKE0001I: The server server has been launched.
[AUDIT ] CWWKF0012I: The server installed the following features: [servlet-3.0].
[AUDIT ] CWWKF0026I: The server server is ready to build a smaller package.
[AUDIT ] CWWKE0036I: The server server stopped after 6.825 seconds.
Building archive for server server.
Server server package complete in C:\IBM\WebSphere\Liberty855_OSGI\usr\servers\server\server.zip.
C:\IBM\WebSphere\Liberty855_OSGI\bin>dir C:\IBM\WebSphere\Liberty855_OSGI\usr\servers\server\server.zip
Volume in drive C has no label.
Volume Serial Number is 3031-6CBE
Directory of C:\IBM\WebSphere\Liberty855_OSGI\usr\servers\server
05/08/2015 03:47 PM          43,945,347 server.zip
                1 File(s)      43,945,347 bytes
                0 Dir(s)      8,905,318,400 bytes free
C:\IBM\WebSphere\Liberty855_OSGI\bin>
C:\IBM\WebSphere\Liberty855_OSGI\bin>server package server
Packaging server server.
Server server package complete in C:\IBM\WebSphere\Liberty855_OSGI\usr\servers\server\server.zip.
C:\IBM\WebSphere\Liberty855_OSGI\bin>dir C:\IBM\WebSphere\Liberty855_OSGI\usr\servers\server\server.zip
Volume in drive C has no label.
Volume Serial Number is 3031-6CBE
Directory of C:\IBM\WebSphere\Liberty855_OSGI\usr\servers\server
05/08/2015 04:03 PM          98,004,525 server.zip
                1 File(s)      98,004,525 bytes
                0 Dir(s)      8,851,329,024 bytes free
C:\IBM\WebSphere\Liberty855_OSGI\bin>
```

Docker support

To ease testing and development cycles, you can also run your application using a pre-built Docker image containing IBM WebSphere Application Server for Developers V8.5.5 Liberty profile and an IBM JAVA Runtime Environment 7.1 in the Docker Hub. This means that you

can get the Liberty profile Docker image environment quickly and run your own applications in there.

For information about Docker Hub, see the following website:

<https://www.docker.com/whatisdocker>

For information about Liberty profile Docker support, see the following website:

http://www-01.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_ca/4/897/ENUS215-084/index.html&request_locale=en

In short, to set your Docker Liberty profile environment, you need to do the following:

1. Set your Docker environment. For information about how to do that on your system, see the following website:

<https://docs.docker.com/installation/#installation>

2. Pull your Liberty profile image using a Docker command, such as in Example 9-14.

Example 9-14 Running a WebSphere Liberty image in Docker

```
docker pull websphere-liberty
Pulling repository websphere-liberty
c659cb20283c: Download complete
e9e06b06e14c: Download complete
a82efea989f9: Download complete
37bea4ee0c81: Download complete
07f8e8c5e660: Download complete
1982456d9ebb: Download complete
068e3636b930: Download complete
16b920ee3a3f: Download complete
2508d7b16997: Download complete
0d361e23268a: Download complete
d6f640fd2033: Download complete
031348e20192: Download complete
71dcc2627667: Download complete
161847a85d6c: Download complete
f5fa021c4927: Download complete
3e165babb523: Download complete
a8a7133801c6: Download complete
c2a2ffe2d254: Download complete
3ce88d18a79e: Download complete
3c650dabf41a: Download complete
6681ce2f90c0: Download complete
9cc0db580156: Download complete
e1972175e2d4: Download complete
Status: Image is up to date for websphere-liberty:latest
```

3. Publish your application to your Docker Liberty profile environment by using a Docker command such as in Example 9-15 on page 254. This command uses the dropins folder of the Liberty profile server to deploy the application. For more information about the dropins directory, see 1.4.1, “Quick start using dropins” on page 28.

Example 9-15 Running an application in a WebSphere Liberty Docker container

```
docker run -d -e LICENSE=accept -p 80:9080 -v  
/tmp/myApp.war:/opt/ibm/wlp/usr/servers/defaultServer/dropins/myApp.war  
websphere-liberty
```

For an example on running an application in a WebSphere Liberty profile of a Docker container, read section 3.3, “Testing with a WebSphere Liberty Docker container” in *Configuring and Deploying Open Source with WebSphere Application Server Liberty Profile*, SG24-8194.

A Liberty product extension is a set of directories and files that have a layout similar to the Liberty product installation but are placed in your own installation location. It can contain the same types that are present in the Liberty product, such as feature and command-line scripts.

The function in a product extension can be fully integrated into the Liberty runtime and WebSphere developer tools. Because it uses the extensible Liberty architecture, it also integrates into future tools that are written to use that extensibility.

A product extension is identified to the Liberty run time through placement of a properties file in the Liberty installation. The name of the properties file is used as the product extension. For convenient development of features, there is a built-in user extension location in the Liberty installation structure: the `usr/extension` directory. You can install your features to that location for easy testing before you package them into your product extension. This is the location that the WebSphere developer tools use for feature installation.

When you design product extensions, there are many aspects that you might want to consider beyond simple function. These design principles are applied to the Liberty product features to keep the servers highly compassable and simple to use. Those design principles are covered in the following sections.

Allowing the user to compose their server

It is a core design principle of the Liberty profile that the user should select which features are in their server. This allows the user to control at a fine-grained level the exact services that are available. There is intentionally no support for profile augmentation, so when your feature is installed into an existing Liberty profile, it is not used by any servers unless the user explicitly adds the feature to the server configuration.

If you want to provide a way for your feature to be used without being manually added to the server configuration, you can provide a server template in the `template servers` directory of your product extension. This approach also allows you to provide a predefined custom configuration in addition to the feature itself. You can direct your users to specify your template on the **server create** command or you can provide your own command script in your extension `bin` directory that wraps the **server create** command and specifies your template, as shown in Example 9-16.

Example 9-16 Wrapping the server create command

```
server create --template=<extension-name>:<template-name>
```

Although this might be a convenient way to allow simple creation of working servers, it should not be used in place of providing useful configuration defaults in your features through metatype. Keeping the default values inside your feature allows you to adjust them in service or future releases. There is no support for modifying existing `server.xml` configuration files. These files are considered to be user-owned files and the Liberty profile does not change them.

For more information about Liberty profile product extensions, see 1.5, “Product extensions” on page 30.

Embedding the Liberty profile server in your applications

You can use the System Programming Interfaces (SPIs) that are provided by the Liberty profile to configure, control, and monitor a Liberty profile server in your applications.

The Liberty profile provides the following two SPIs to start or stop a Liberty profile server:

- ▶ `com.ibm.wsspi.kernel.embeddable.Server`
- ▶ `com.ibm.wsspi.kernel.embeddable.ServerBuilder`

Use a **Future** object to store the result of a start or stop operation. The return codes that are used by embedded operations are the same as the return codes used by the server command. Additionally, you can receive asynchronous notifications when the server is starting, has started, or has stopped by creating your own class that implements the `com.ibm.wsspi.kernel.embeddable.ServerEventListener` interface.

To include your Liberty profile server within your application, you must follow these steps:

1. Add to the class path the `ws-server.jar` file, which is in the `${wlp.install.dir}/bin/tools` directory of the Liberty profile installation.
2. Specify the name of your target server.

Notes:

- ▶ Environment variables are not checked and the `jvm.options` and `server.env` files are not read.
- ▶ Management of the JVM and environment is assumed to be managed by the caller.
- ▶ Configure the `ws-javaagent.jar` file with the `-javaagent` JVM option. The `ws-javaagent.jar` file is in the `${wlp.install.dir}/bin/tools` directory of the Liberty profile installation. You are advised to configure the `ws-javaagent.jar` file, but it is not mandatory unless you use capabilities of the server that require it, such as monitoring or trace. If you contact IBM support, you might need to provide trace, and if so, you must start the server with the `ws-javaagent.jar` file, even if you do not normally use it.

For more information about procedure to embed your Liberty profile server into your applications, see the following website:

http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/twlp_extend_embed.html

All configuration is specified in the server.xml file

Almost all configuration for a server can be specified in the `server.xml` or other included configuration files. One exception is the case where properties must be set before `server.xml` processing. For example, JVM properties cannot be set in `server.xml` and for those you can use the `bootstrap.properties` or `jvm.options` files. Another exception is the case where a public specification requires that the configuration be contained in a specific file. Specifying configuration in the `server.xml` or included files allows an ecosystem of commands and tools to be built on those files.

By receiving your configuration from the `server.xml` file, you benefit from the following capabilities:

- ▶ Automatic parsing and injection of your configuration properties

- ▶ Reinjection every time the property values are updated
- ▶ Inclusion of your configuration in the generated schema that is used by the WebSphere developer tools and any future graphical user interfaces
- ▶ Management of configuration files by the Liberty run time

It also benefits your users by allowing them to keep all of their server configurations in a single file or set of files.

Keeping it small and fast

Keep your dependency stack as short as possible. Defer as much of your initialization processing as possible. For example, you can use OSGi Declarative Services to delay loading of your service implementation classes until they are used.

Dynamic behavior

Features should be able to stop and restart independent of the server restarting and independent of each other. This consideration relies on correct cleanup on shutdown and the dynamic management of references to services in other features. Your features should react dynamically to configuration updates. This dramatically reduces the need for servers to be restarted, especially during application development. It also greatly increases usability.

Simple migration

It should be painless for users to apply service and product upgrades. Some design points that help to achieve that goal are included in the following list:

- ▶ No migration is required for configuration; all configuration remains compatible with earlier versions.
- ▶ Longer toleration of Java versions.
- ▶ Clear separation of product and user files and internal and external function.
- ▶ Feature evolution is strictly compatible with earlier versions. The versions in feature names provide a way to handle incompatibilities in upgrades by providing a new feature version with the new behavior but continuing to support the existing feature.



A

Additional material

This book refers to additional material that can be downloaded from the Internet, as described in the following sections.

Locating the web material

The web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser to the following URL:

<ftp://www.redbooks.ibm.com/redbooks/SG248076>

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248076.

Using the web material

The additional web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
ch3_ITSOWeb.zip	A Liberty profile developer tools project with code samples from Chapter 3, “Developing and deploying web applications” on page 59
ch4_OSGiExample.zip	A Liberty profile developer tools project with code samples from Chapter 4, “Iterative development of OSGi applications” on page 99
ch5_ITSOWeb.zip	A Liberty profile developer tools project with code samples from Chapter 5, “Developing enterprise applications with Liberty profile” on page 113

ITSO_derby.zip	A Derby database for use with samples from Chapter 5, “Developing enterprise applications with Liberty profile” on page 113
ITSO2_derby.zip	A Derby database for use with samples from Chapter 5, “Developing enterprise applications with Liberty profile” on page 113
ITSODD_derby.zip	A Derby database for use with samples from Chapter 5, “Developing enterprise applications with Liberty profile” on page 113

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and download the contents of the web material files into this folder. Extract the `sg248076.zip` file into this folder to access the sample application files.

Importing the Liberty profile developer tools projects

To use any of the Liberty profile developer tools projects, you must first create a Liberty profile server runtime environment on your Eclipse installation, as described in 2.2.1, “Installation using the Liberty profile developer tools” on page 43.

Use the following process to use any of the three example projects: `ch3_ITSOWeb.zip`, `ch4_OSGiExample.zip`, or `ch5_ITSOWeb.zip`:

1. Extract the compressed project file into its own subdirectory.
2. From Eclipse, run **File** → **Import** → **Existing Projects Into Workspace**.
3. In the Import Projects window, enter the path to the subdirectory where you extracted the compressed file in the Select root directory field. This should complete the Projects field with either ITSOWeb (for the examples from Chapter 3, “Developing and deploying web applications” on page 59 and Chapter 5, “Developing enterprise applications with Liberty profile” on page 113), or five projects beginning with ITSO.OSGiExample (for the examples from Chapter 4, “Iterative development of OSGi applications” on page 99).
4. Select the box to copy projects into the workspace.
5. Click **Finish**.

Using the compressed Derby database files

To use the compressed Derby database files, simply extract the compressed files and make a note of the file path. The examples in Chapter 5, “Developing enterprise applications with Liberty profile” on page 113 describe how to set up a data source to point to these example databases.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

Online resources

These websites are relevant as further information sources:

- ▶ Enterprise Bundle Archive (EBA) Maven Plugin:
<http://aries.apache.org/modules/ebamavenpluginproject.html>
- ▶ OSGi home page:
<http://www.osgi.org>
- ▶ Stack Overflow tags:
<http://stackoverflow.com/questions/tagged/websphere-liberty>
- ▶ Supported operating systems for the Liberty profile:
<http://www.ibm.com/support/docview.wss?uid=swg27038218>
- ▶ Technologies that are supported by the Liberty profile:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_prog_model_support.html?cp=SSAW57_8.5.5%2F1-0-2-0-0
- ▶ WASdev community:
<http://wasdev.net>
- ▶ WASdev forum:
<https://developer.ibm.com/wasdev>
- ▶ WebSphere Application Server Developer Tools for Eclipse:
http://publib.boulder.ibm.com/infocenter/radhelp/v9/topic/com.ibm.rad.install.doc/topics/t_install_wdt.html
- ▶ WebSphere Application Server Liberty Profile IBM Knowledge Center:
http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/welc6tech_wlp_thr.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

IBM WebSphere Application Server Liberty Profile Guide for Developers

SG24-8076-02

ISBN 0738440876



(0.5" spine)

0.475" x 0.873"

250 <-> 459 pages



SG24-8076-02

ISBN 0738440876

Printed in U.S.A.

Get connected

