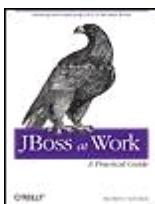


NEXT ➔**JBoss at Work: A Practical Guide**

By Scott Davis, Tom Morris

Publisher: **O'Reilly**Pub Date: **October 2005**ISBN: **0-596-00734-5**Pages: **306**[Table of Contents](#) [Index](#)

Overview

Consisting of a number of well-known open source products, JBoss is more a family of interrelated services than a single monolithic application. But, as with any tool that's as feature-rich as JBoss, there are number of pitfalls and complexities, too.

Most developers struggle with the same issues when deploying J2EE applications on JBoss: they have trouble getting the many J2EE and JBoss deployment descriptors to work together; they have difficulty finding out how to get started; their projects don't have a packaging and deployment strategy that grows with the application; or, they find the Class Loaders confusing and don't know how to use them, which can cause problems.

JBoss at Work: A Practical Guide helps developers overcome these challenges. As you work through the book, you'll build a project using extensive code examples. You'll delve into all the major facets of J2EE application deployment on JBoss, including JSPs, Servlets, EJBs, JMS, JNDI, web services, JavaMail, JDBC, and Hibernate. With the help of this book, you'll:

- Implement a full J2EE application and deploy it on JBoss
- Discover how to use the latest features of JBoss 4 and J2EE 1.4, including J2EE-compliant web services
- Master J2EE application deployment on JBoss with EARs, WARs, and EJB JARs
- Understand the core J2EE deployment descriptors and how they integrate with JBoss-specific descriptors
- Base your security strategy on JAAS

Written for Java developers who want to use JBoss on their projects, the book covers the gamut of deploying J2EE technologies on JBoss, providing a brief survey of each subject aimed at the working professional with limited time.

If you're one of the legions of developers who have decided to give JBoss a try, then *JBoss at Work: A Practical Guide* is your next logical purchase. It'll show you in plain language how to use the fastest growing open source tool in the industry today. If you've worked with JBoss before, this book will get you up to speed on JBoss 4, JBoss WS (web services), and Hibernate 3.

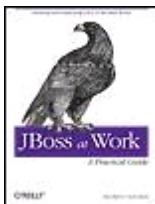
NEXT ➔



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)**JBoss at Work: A Practical Guide**

By Scott Davis, Tom Farrs

Publisher: **O'Reilly**
 Pub Date: **October 2005**
 ISBN: **0-596-00734-5**
 Pages: **306**

[Table of Contents](#) [Index](#)

- ? [Copyright](#)
- ? [About the Author](#)
- ? [Preface](#)
- ? ? [Audience](#)
- ? ? [About This Book](#)
- ? ? [Assumptions This Book Makes](#)
- ? ? [Conventions Used in This Book](#)
- ? ? [Using Code Examples](#)
- ? ? [Safari Enabled](#)
- ? ? [Comments and Questions](#)
- ? ? [Acknowledgments](#)
- ? ? [Chapter 1. Getting Started with JBoss](#)
- ? ? [Section 1.1. Why "JBoss at Work"?](#)
- ? ? [Section 1.2. Why JBoss?](#)
- ? ? [Section 1.3. The Example: JAW Motors](#)
- ? ? [Section 1.4. The Tools](#)
- ? ? [Section 1.5. Installing JBoss](#)
- ? ? [Section 1.6. Deploying Applications to JBoss](#)
- ? ? [Section 1.7. Looking Ahead...](#)
- ? ? [Chapter 2. Web Applications](#)
- ? ? [Section 2.1. The Servlet Container](#)
- ? ? [Section 2.2. Three-Tier Applications](#)
- ? ? [Section 2.3. Exploring the Presentation Tier](#)
- ? ? [Section 2.4. Building the View_Cars Page](#)
- ? ? [Section 2.5. Adding a Model and Controller](#)
- ? ? [Section 2.6. Looking Ahead...](#)
- ? ? [Chapter 3. Building and Deploying an FAR](#)
- ? ? [Section 3.1. WARS Versus FARs](#)
- ? ? [Section 3.2. Application.xml](#)
- ? ? [Section 3.3. Common JAR](#)
- ? ? [Section 3.4. Deploying the FAR](#)
- ? ? [Section 3.5. Adding a DAO](#)
- ? ? [Section 3.6. Using XDoclet](#)
- ? ? [Section 3.7. Looking Ahead...](#)
- ? ? [Chapter 4. Databases and JBoss](#)
- ? ? [Section 4.1. Persistence Options](#)
- ? ? [Section 4.2. JDBC](#)
- ? ? [Section 4.3. JNDI](#)
- ? ? [Section 4.4. JNDI References in web.xml](#)
- ? ? [Section 4.5. JBoss DataSource Descriptors](#)
- ? ? [Section 4.6. JDBC Driver JARs](#)
- ? ? [Section 4.7. Database Checklist](#)
- ? ? [Section 4.8. Accessing the Database Using Ant](#)
- ? ? [Section 4.9. Creating JDBC CarDAO](#)
- ? ? [Section 4.10. Looking Ahead...](#)

- ? ? [Chapter 5.?Hibernate and JBoss](#)
- ? ? [Section 5.1.?The Pros and Cons of ORMs](#)
- ? ? [Section 5.2.?Hibernate Mapping Files](#)
- ? ? [Section 5.3.?Hibernate MBean Service Descriptor](#)
- ? ? [Section 5.4.?Creating a HAR](#)
- ? ? [Section 5.5.?Adding the HAR to the EAR](#)
- ? ? [Section 5.6.?Creating a JNDI Lookup](#)
- ? ? [Section 5.7.?Hibernate Checklist](#)
- ? ? [Section 5.8.?HibernateCarDAO](#)
- ? ? [Section 5.9.?Adding a Car](#)
- ? ? [Section 5.10.?Editing a Car](#)
- ? ? [Section 5.11.?Deleting a Car](#)
- ? ? [Section 5.12.?Looking Ahead...](#)
- ? ? [Chapter 6.?Stateless Session Beans](#)
- ? ? [Section 6.1.?Issues with EJBs](#)
- ? ? [Section 6.2.?Should I Use EJB or Not?](#)
- ? ? [Section 6.3.?Business Tier](#)
- ? ? [Section 6.4.?Enterprise JavaBeans](#)
- ? ? [Section 6.5.?Our Example](#)
- ? ? [Section 6.6.?Iteration 1Introduce a Session Bean](#)
- ? ? [Section 6.7.?Calling the Session Bean from the Controller Servlet](#)
- ? ? [Section 6.8.?EJB-Based JNDI References in Web-Based Deployment Descriptors](#)
- ? ? [Section 6.9.?Session Bean Types](#)
- ? ? [Section 6.10.?Session Beans](#)
- ? ? [Section 6.11.?Remote Versus Local EJB Calls](#)
- ? ? [Section 6.12.?Local and Remote Interfaces](#)
- ? ? [Section 6.13.?Home Interfaces](#)
- ? ? [Section 6.14.?Review ing Iteration 1](#)
- ? ? [Section 6.15.?Testing Iteration 1](#)
- ? ? [Section 6.16.?Iteration 2Move Business Logic Out of the Controller](#)
- ? ? [Section 6.17.?Review ing Iteration 2](#)
- ? ? [Section 6.18.?Testing Iteration 2](#)
- ? ? [Section 6.19.?Iteration 3Buy a Car](#)
- ? ? [Section 6.20.?The AccountingDTO](#)
- ? ? [Section 6.21.?Developing the HibernateAccountingDAO](#)
- ? ? [Section 6.22.?Adding buyCar\(\) to the InventoryFacadeBean](#)
- ? ? [Section 6.23.?Review ing Iteration 3](#)
- ? ? [Section 6.24.?Testing Iteration 3](#)
- ? ? [Section 6.25.?Final Thoughts on Session Beans](#)
- ? ? [Section 6.26.?Looking Ahead ...](#)
- ? ? [Chapter 7.?Java Message Service \(JMS\) and Message-Driven Beans](#)
- ? ? [Section 7.1.?Sending Messages with JMS](#)
- ? ? [Section 7.2.?Upgrade the Site: Running a Credit Check](#)
- ? ? [Section 7.3.?JMS Architecture Overview](#)
- ? ? [Section 7.4.?JMS Messaging Models](#)
- ? ? [Section 7.5.?Creating a Message](#)
- ? ? [Section 7.6.?Sending the Message](#)
- ? ? [Section 7.7.?Core JMS API](#)
- ? ? [Section 7.8.?Sending a JMS Message](#)
- ? ? [Section 7.9.?JMS-Based JNDI References in Web-Based Deployment Descriptors](#)
- ? ? [Section 7.10.?Deploying JMS Destinations on JBoss](#)
- ? ? [Section 7.11.?JMS Checklist](#)
- ? ? [Section 7.12.?Message-Driven Beans \(MDBs\)](#)
- ? ? [Section 7.13.?MDB Checklist](#)
- ? ? [Section 7.14.?Testing the Credit Check](#)
- ? ? [Section 7.15.?Looking Ahead ...](#)
- ? ? [Chapter 8.?JavaMail](#)

- ? ? [Section 8.1.?Running a Credit Check](#)
- ? ? [Section 8.2.?Sending Email Messages w ith JavaMail](#)
- ? ? [Section 8.3.?Upgrading the MDB to Send an Email Message](#)
- ? ? [Section 8.4.?Sending an Email Message](#)
- ? ? [Section 8.5.?JavaMail-Based JNDI References in EJB Deployment Descriptors](#)
- ? ? [Section 8.6.?Automating JavaMail-Based JNDI References w ith XDoclet](#)
- ? ? [Section 8.7.?Deploying JavaMail on JBoss](#)
- ? ? [Section 8.8.?JavaMail Checklist](#)
- ? ? [Section 8.9.?Testing the Credit Check Notification Email](#)
- ? ? [Section 8.10.?Looking Ahead ...](#)
- ? ? [Chapter 9.?Security](#)
 - ? ? [Section 9.1.?J2EE Security](#)
 - ? ? [Section 9.2.?Web-Based Security](#)
 - ? ? [Section 9.3.?Restricting Access w ith web.xml](#)
 - ? ? [Section 9.4.?JAAS](#)
 - ? ? [Section 9.5.?Deploying a JAAS-Based Security Realm on JBoss](#)
 - ? ? [Section 9.6.?Testing Secure JSPs](#)
 - ? ? [Section 9.7.?Protecting the Administrative Actions](#)
 - ? ? [Section 9.8.?Web Security Checklist](#)
 - ? ? [Section 9.9.?Integrating Web Tier and EJB Tier Security](#)
 - ? ? [Section 9.10.?EJB Security](#)
 - ? ? [Section 9.11.?EJB Security Checklist](#)
 - ? ? [Section 9.12.?Looking Ahead ...](#)
- ? ? [Chapter 10.?Web Services](#)
 - ? ? [Section 10.1.?Web Services Architecture](#)
 - ? ? [Section 10.2.?JBoss 4.x and Web Services](#)
 - ? ? [Section 10.3.?J2EE 1.4 and Web Services](#)
 - ? ? [Section 10.4.?Implementing J2EE 1.4 Web Services](#)
 - ? ? [Section 10.5.?Service Endpoint Interface \(SEI\)](#)
 - ? ? [Section 10.6.?Modifying ejb-jar.xml](#)
 - ? ? [Section 10.7.?webservices.xml](#)
 - ? ? [Section 10.8.?JAX-RPC Mapping File](#)
 - ? ? [Section 10.9.?WSDL File](#)
 - ? ? [Section 10.10.?Set the Web Service URL](#)
 - ? ? [Section 10.11.?Modifying the InventoryFacadeBean EJB](#)
 - ? ? [Section 10.12.?Web Services Deployment](#)
 - ? ? [Section 10.13.?Automating Web Services Deployment](#)
 - ? ? [Section 10.14.?J2EE Web Services Checklist](#)
 - ? ? [Section 10.15.?Testing Web Services Deployment](#)
 - ? ? [Section 10.16.?Web Services Client](#)
 - ? ? [Section 10.17.?Implementing a Web Service Client](#)
 - ? ? [Section 10.18.?Web Service Client Checklist](#)
 - ? ? [Section 10.19.?Testing the Web Service Client](#)
 - ? ? [Section 10.20.?Final Thoughts on J2EE 1.4 Web Services](#)
 - ? ? [Section 10.21.?Conclusion](#)
 - ? ? [Section 10.22.?Congratulations!](#)
- ? ? [Appendix A.?ClassLoaders and JBoss](#)
 - ? ? [Section A.1.?Namespaces](#)
 - ? ? [Section A.2.?Class Loading in the J2EE](#)
 - ? ? [Section A.3.?Class Loading w ith JBoss](#)
 - ? ? [Section A.4.?Common ClassLoader Issues](#)
 - ? ? [Section A.5.?ClassLoader Options](#)
 - ? ? [Section A.6.?Solving ClassLoader Issues](#)
 - ? ? [Section A.7.?Conclusion](#)
- ? ? [Appendix B.?Logging and JBoss](#)
 - ? ? [Section B.1.?Jakarta Commons Logging \(JCL\) API](#)
 - ? ? [Section B.2.?Apache Log4J](#)

- ? ? [Section B.3.?Adding Application-Specific Properties to System Properties](#)
- ? ? [Section B.4.?Configuring Log4J with a Configuration File](#)
- ? ? [Section B.5.?Loading Resources from the CLASSPATH](#)
- ? ? [Section B.6.?Logging Deployment](#)
- ? ? [Section B.7.?Logging Checklist](#)
- ? ? [Section B.8.?Testing Logging](#)
- ? ? [Section B.9.?Conclusion](#)
- ? ? [Appendix C.?JAAS Tutorial](#)
- ? ? [Section C.1.?JAAS](#)
- ? ? [Section C.2.?Client-Side JAAS](#)
- ? ? [Section C.3.?Conclusion](#)
- ? [Colophon](#)
- ? [Index](#)

◀ PREV

NEXT ▶



[◀ PREV](#)

[NEXT ▶](#)

JBoss at Work: A Practical Guide

by Tom Marrs and Scott Davis

Copyright ?2006 O'Reilly Media, Inc. All rights reserved. Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor:	Mike Loukides
Production Editor:	Colleen Gorman
Cover Designer:	Karen Montgomery
Interior Designer:	David Futato

Printing History:

October 2005:	First Edition.
---------------	----------------

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *JBoss?at Work: A Practical Guide*, the image of a golden eagle, and related trade dress are trademarks of O'Reilly Media, Inc.

Java?and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. O'Reilly Media, Inc. is independent of Sun Microsystems.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 0-596-00734-5

[M]

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

About the Author

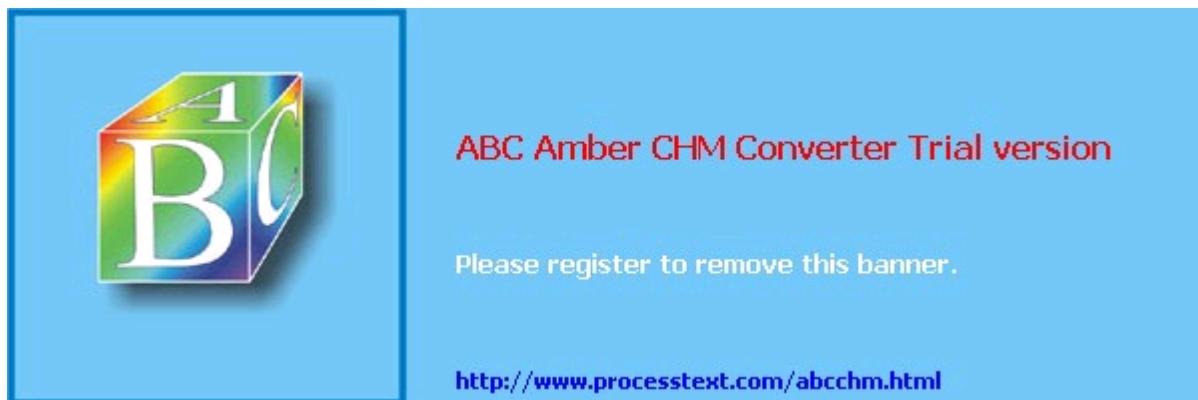
Tom Marrs, a 20-year veteran in the software industry, is the Principal and Senior Software Architect at Vertical Slice, a consulting firm that designs and implements mission-critical business applications using the latest J2EE and open source technologies, along with providing architecture evaluation and developer training and mentoring services. Tom teaches Java/J2EE/JBoss training classes, speaks regularly at software conferences such as No Fluff Just Stuff (<http://www.nofluffjuststuff.com>), and is a blogger on [java.net](#) and ONJava. An active participant in the local technical community, Tom is the President of the Denver JBoss User Group and has served as President of the Denver Java Users Group (<http://www.denverjug.org>).

Scott Davis is a senior software engineer in the Colorado front range. He is passionate about open source solutions and agile development. He has worked on a variety of Java platforms, from J2EE to J2SE to J2ME (sometimes all on the same project).

Scott is a frequent presenter at national conferences and local user groups. He was the president of the Denver Java Users Group in 2003 when it was voted one of the Top 10 JUGs in North America. After a quick move north, he is currently active in the leadership of the Boulder Java Users Group. Keep up with him at <http://www.davisworld.org>.

◀ PREV

NEXT ▶



 PREV

NEXT 

Preface

Are you curious about the cool, new features of JBoss 4 and J2EE 1.4? Are you frustrated with all the simplistic "Hello World" examples? Do you want to see a realistic application deployed on JBoss?

As practitioners, we've seen that most people struggle with the following issues when deploying J2EE applications on JBoss:

- Real application deployment involves many J2EE and JBoss deployment descriptors, and it's difficult to make them all work together.
- Developers new to JBoss need a way to get started.
- Most projects don't have a packaging and deployment strategy that grows with their application.
- Class Loaders are confusing and can cause problems when developers don't know how to use them.

This book shows you how to use JBoss with the latest Open Source Java tools. By building a project throughout the book with extensive code examples, we cover all major facets of J2EE application deployment on JBoss, including JSPs, Servlets, EJBs, JMS, JNDI, Web Services, JavaMail, JDBC, and Hibernate.

With the help of this book, you'll:

- Implement a full J2EE application and deploy it on JBoss.
- Discover how to use the latest features of JBoss 4 and J2EE 1.4, including J2EE-compliant Web Services.
- Master J2EE application deployment on JBoss with EARs, WARs, and EJB JARs.
- Understand the core J2EE deployment descriptors and how they integrate with JBoss-specific descriptors.
- Deploy JSPs, Servlets, EJBs, JMS, Web Services, JavaMail, JDBC, and Hibernate on JBoss.
- Base your security strategy on JAAS.

Although this book covers the gamut of deploying J2EE technologies on JBoss, it isn't an exhaustive discussion of each aspect of the J2EE API. This book is meant to be a brief survey of each subject aimed at the working professional with limited time.

 PREV

NEXT 



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

Audience

This book is for Java developers who want to use JBoss on their projects. If you're new to J2EE, this book will serve as a gentle introduction. But don't mistake this book for a true J2EE reference manual. There is a reason those books are 1,000+ pages longthey cover each technology in exhaustive detail. This book gives you enough to get a simple example up and running quickly.

If you've worked on J2EE projects but are new to JBoss, this book covers familiar concepts, introduces you to key J2EE 1.4 issues including Web Services, and shows you how to make them work with JBoss. If you've worked with JBoss before, this book will get you up to speed on JBoss 4, JBoss WS (Web Services), and Hibernate 3.

◀ PREV

NEXT ▶



[◀ PREV](#)

[NEXT ▶](#)

About This Book

This book starts with a simple web page and iteratively shows you how to add the various J2EE technologies to develop an application that runs on JBoss. Rather than getting stuck in the details of every possible J2EE API or J2EE/JBoss deployment descriptor, we focus on learning by doing. We introduce you to each topic, show what we're going to do, do it, and recap what we did. By taking an iterative approach, we keep things short, sweet, and to the point so that you can put JBoss to work on your projects.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

Assumptions This Book Makes

We assume that you've worked with Java and are familiar with Open Source tools such as Ant and XDoclet. We show you how to download and install them. We provide you with Ant scripts for compiling and deploying the application.

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

Conventions Used in This Book

The following typographical conventions are used in this book:

Plain text

Indicates menu titles, menu options, menu buttons, and keyboard accelerators (such as Alt and Ctrl).

Italic

Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities.

Constant width

Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values.



This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*JBoss at Work: A Practical Guide*, by Tom Marrs and Scott Davis. Copyright 2005 O'Reilly Media, Inc., 0-596-00734-5."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

Safari Enabled

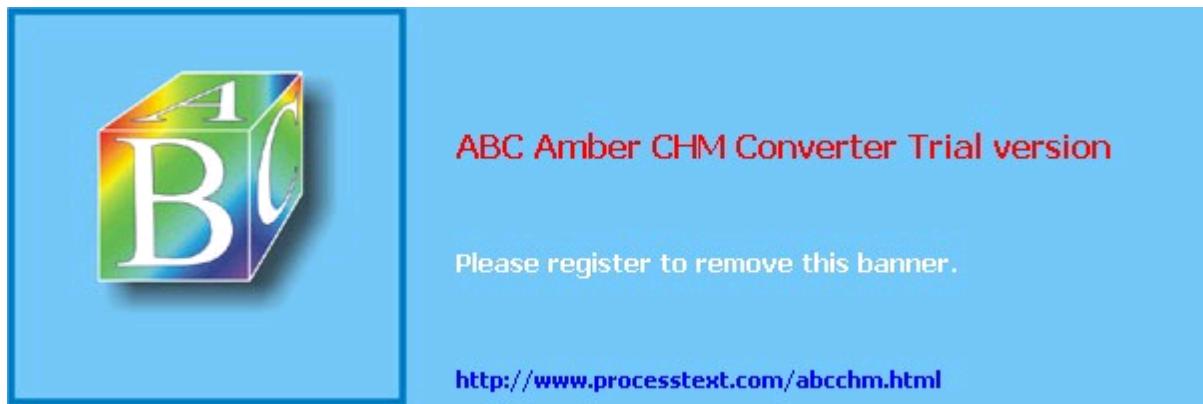


When you see a Safari?enabled icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at <http://safari.oreilly.com>.

◀ PREV

NEXT ▶

A blue rectangular banner with a white border. On the left side, there is a 3D-style cube with a large white letter 'B' on the front face and a smaller '1' on the top face. To the right of the cube, the text "ABC Amber CHM Converter Trial version" is displayed in red. Below this, in white text, is the instruction "Please register to remove this banner." At the bottom right of the banner, the URL "<http://www.processtext.com/abcchm.html>" is shown in blue.

ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

<http://www.oreilly.com/catalog/jbossatwork>

We also have a companion web site for this book, where we have an FAQ, links to resources, and bonus materials. Please visit:

<http://www.jbossatwork.com>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com
tom@jbossatwork.com
scott@jbossatwork.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

<http://www.oreilly.com>

◀ PREV

NEXT ▶



 PREV

NEXT 

Acknowledgments

Many people contributed to this book's development. We're grateful to Mike Loukides, our editor, for his experience, guidance, and direction. We'd like to thank him for believing in us and being patient with two first-time authors as we learned our craft.

We had a great team of expert technical reviewers who helped ensure sure that the material was technically accurate, approachable, and reflected the spirit of the JBoss, J2EE, and Open Source communities. Our reviewers were Norman Richards, Greg Ostravich, Andy Ochsner, and Dan Moore. Their suggestions and corrections greatly improved the quality of the book. We're especially thankful to Norman Richards of JBoss, Inc. for his quick turnaround on show-stopper issues and for all his great advice.

We owe a great debt to the Open Source community who made the tools for this book:

- To JBoss, Inc. for creating and maintaining JBoss, an outstanding and reliable J2EE application server that we use on our jobs every day. JBoss is great and we love it. We hope that the concept of Professional Open Source will continue to blossom and grow.
- To the Ant, XDoclet, Log4J, Apache Jakarta, Hibernate, and (numerous) Apache and SourceForge projectsyou guys rock! Your tools keep the Java community going.

Tom's Acknowledgments

I am especially thankful to Scott Davis, my co-author, for exhorting me to finish the book, holding me accountable, and for pushing me to improve my writing style. This book would've been impossible without him.

Thanks to Richard Monson-Haefel, Sue Spielman, Bruce Tate, Brett McLaughlin, Frank Tradition (my business coach), the Denver Java Users Group (DJUG<http://www.denverjug.org>), and everyone else who encouraged me along the way.

Thanks to Jay Zimmerman, coordinator of the "No Fluff Just Stuff" (<http://www.nofluffjuststuff.com>) conferences, for enabling me to take my message on the road.

Thanks to The One Way Caf?in Aurora, COkeep the lattes and good advice flowing.

Most importantly, I am deeply grateful to my wife, Linda, and daughter, Abby, for supporting me during the writing process. I love you and look forward to spending more time together.

Scott's Acknowledgments

Tom came to me with an opportunity to co-author a book for O'Reilly. How could I possibly turn down a gig like that? Tom and I have known each other for years, and we knew from the start that we brought complimentary skills to the table. This book was a collaborative effort in every sense of the word, but it never would have happened if Tom hadn't planted the first seed.

What started out as a wildly optimistic (and in retrospect, totally unrealistic) attempt to map out the entire known world of Open Source J2EE development eventually got distilled down to the book you are now holding. Even though this book is far more modest in scope than our original idea, I think that it still captures the spirit of what we set out to accomplish. Without getting bogged down in the whole commercial versus free versus open source quagmire, we wanted to show you that it is possible to create a production-quality application using nothing but freely available tools.

Thanks go out to the Denver and Boulder JUG communitieschanging out with all of you (too numerous to mention individually) has made me a better programmer and a better person. When I was a lone wolf contractor, your emails and IMs, phone calls and lunches, but especially the post-meeting pints and horror story-swaps are what kept me sane through all of it. When I was new to a city and a programming language, you made me feel like I belonged.

A very warm thanks goes out to Jay and the whole NFJS crew (Ted, Bruce, Erik, Jason, James, Mike, Stu, Justin, Glenn, David, Eitan, Dian, Ben, Dave, and the rest of ya'll). After

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)

Chapter 1. Getting Started with JBoss

Have you noticed that simply saying "I am a Java programmer" isn't enough these days? It conveys a little bit of information, but not enough to make any serious decisions. It's kind of like saying, "I play sports" or "I like food." A recruiter can assume that a Java programmer has a passing familiarity with curly braces and semicolons, but little else.

The Java programming language runs on an incredibly diverse set of hardware from cell phones and PDAs down to embedded chips on a credit card; every major desktop and laptop, regardless of operating system or hardware manufacturer; entry-level workgroup servers up to clusters of high-end servers; and even mainframes.

The mantra in the heady early days of Java was, "Write once, run anywhere." The original ideal of having the same application run anywhere from a cell phone to a large-scale cluster of servers turned out to be more marketing hype than business reality, although the "run anywhere" part of the slogan has proven remarkably prescient.

Modern Java developers often define themselves by the hardware they specialize in. J2ME developers eke amazing functionality out of resource-starved micro-devices with limited networking capabilities. J2SE programmers have mastered daunting but robust GUI frameworks such as Swing and SWT for rich desktop application development. And J2EE software engineers are masters of the server-side domain.

Saying that you are a J2EE programmer begins to narrow the field a bit, but our hypothetical recruiter still doesn't have enough information to place you in the proper job. J2EE is a loose collection of server-side technologies that are related, but are by no means homogenous.

Some J2EE experts specialize in web-related technologies JSPs, Servlets, and the diverse landscape of web frameworks such as Jakarta Struts or Sun's Java Server Faces. Others are back-end specialists that focus more on the transactional integrity and reliability of business processing that uses technologies such as EJBs, JMS, and relational databases. (We'll define these acronyms later in the book.) There is even a new breed of Web Services specialists that use the J2EE product suite and a host of related XML technologies, such as SOAP and WSDL, to offer a Service Oriented Architecture to Java and non-Java clients alike.

Asking any one specialist to describe the J2EE toolkit brings to mind the story of the blind men and the elephant. Each blind man describes the elephant based on the part he touches—the one holding the trunk describes a very different animal than the one holding the tusk or the ear.

This book attempts to describe the whole elephant in the context of JBoss, an open source J2EE container. Like the technology it implements, JBoss is not a single monolithic application. Rather, it is a family of interrelated services that correspond to each item in the J2EE collection.

Each chapter in this book explores one of the J2EE services, but unlike the blind men, we show how one technology works in conjunction with the others. A J2EE application is often greater than the sum of its parts, and understanding the J2EE collection means understanding how each piece is interrelated.

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

1.1. Why "JBoss at Work"?

Before we get too far into things, we should explain why we chose the title *JBoss at Work* for this book. Understanding the authors' backgrounds should help.

Both of us are practicing software engineers who have worked together off and on for years. More importantly, both of us are former presidents of the Denver Java Users Group (<http://www.denverjug.org>). When we polled the group for potential interest in a given subject, the same phrase came up over and over again: "I don't want to be an expert in it, I just want to make it work."

"I just want to make it work" really resonates with us because we feel the same way. An ever-growing number of technologies fall under the J2EE umbrella, and there are at least two or three competing implementations of each. Just trying to keep up is a never-ending battle.

There is a 1,000-page book out there for each topic we cover in only 20 to 30 pages. *JBoss At Work* isn't intended to be an exhaustive discussion of every facet of the J2EE collection. This book is meant to be a brief survey of each subject aimed at the working professional with limited time "Give me an overview, show me some working code, and make it snappy...." (Think of it as 12 months of JUG presentations collected in a single volume, minus the PowerPoint slides and cold pizza.)

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

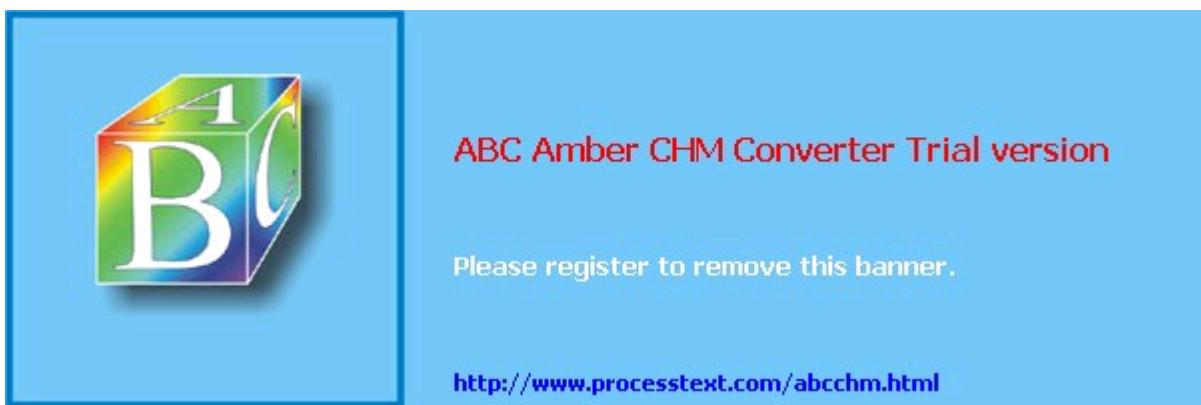
1.2. Why JBoss?

JBoss fits the "I just want to make it work" gestalt to a T. Depending on the speed of your Internet connection, you can have it downloaded, unzipped, and running in less than five minutes. Turning services on and off is as simple as adding or removing files from a directory. The fact that it's free means that you don't get bogged down with per-seat or per-CPU licensing costs. JBoss is both a great learning tool and a production-quality deployment environment.

But any tool as powerful as JBoss also has pitfalls and complexities. The biggest disservice we could do is show you how to write applications that are tied to a specific application server, JBoss or otherwise. The "Write Once, Run Anywhere" promise of J2EE development may not happen automatically, but you can take steps to minimize the impact of moving from one application server to the next. In addition to your code being more portable, being a non-partisan J2EE developer means that *you and your skills* are more portable as you move from one job to the next.

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

1.3. The Example: JAW Motors

We have tried to come up with an application that uses each layer of the J2EE collection in some sort of meaningful way. By design, this book is short on academic discussions and long on working code examples. Showing a coherent business application in action will hopefully give you a clearer idea of how the various layers interact, as opposed to a series of disjointed "Hello World" examples exercising each layer in isolation.

The JAW Motors application supports a fictitious automobile dealership. Each chapter progressively adds a new J2EE technology that solves a specific business problem. Viewing cars on a website involves JSP pages and some form of persistence (JDBC or Hibernate). Performing a credit check sends a JMS message and an email response using JavaMail. Purchasing a car requires the transactional support of Stateless Session Beans. Sharing data from the JAW Motors inventory with other dealerships involves setting up Web Services.

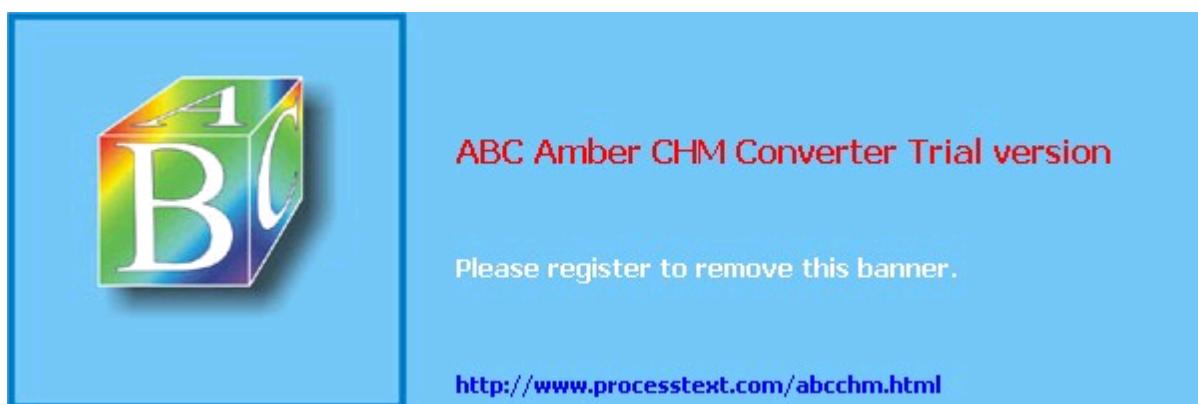
In addition to showing how JBoss works, we hope that these examples answer the how and why of each technology: how is it used, and why it should (or shouldn't) be used. Just because a hammer can sink a screw in drywall doesn't necessarily mean that it is the best tool for the job. The measure of a successful J2EE application isn't how many of the technologies it uses; it is how effectively each technology is used.

Source code for the JAW Motors application is available for download from <http://www.jbossatwork.com>. We encourage you to download the files and build them as you follow along in the book. We want you to literally see *JBoss at work*, not just read about it.

Before we get too far, let's make sure that you have all necessary tools to build and deploy the application to JBoss.

◀ PREV

NEXT ▶



 PREV

NEXT 

1.4. The Tools

Making JBoss work involves more than just downloading and running JBoss. A cook certainly needs to know how to run the oven, but a lot of preliminary work must happen before the dish is ready for baking.

Professional chefs call this set up process "mis en place." They sharpen knives and place cutting boards within arms' reach. They prepare ahead of time ingredients they can safely cut up and measure before the dinner rush. Everything that can be done in terms of efficiency is handled up front so the culinary artist isn't distracted by mundane details.

Similarly, making JBoss work effectively requires you to do a bunch of work up front. Code must be compiled and packaged up in a specific way. You must wade through endless deployment descriptors. If one tiny piece of information doesn't match up with its companion in another file, the application will not deploy properly, and all of your hard work will be for nothing.

The mis en place of JBoss development involves other tools that make it easy to handle the mundane details of building and deploying your application. As in JBoss, you can download and use all of these tools for free:

- Java
- Ant
- XDoclet

Let's talk briefly about how to install and configure them.

1.4.1. Installing Java

It probably goes without saying that the first thing you'll need is a working installation of Java. JBoss 4.0.2 is compatible with J2SE 1.4 or higher. We use J2SE 1.4.2 in this book, although nothing should prevent you from running the examples in Java 5.

Download the full JDK (Java 2 Development Kit) from Sun's web site (<http://java.sun.com>). Follow Sun's instructions for installing the JDK on your operating system. Next, create an environment variable called *JAVA_HOME* that points to the Java installation directory. Finally, add *\$JAVA_HOME/bin* to the system path so you can run Java from the command line.

To verify your Java installation, type `java -version` at a command prompt. You should see [Example 1-1](#).

Example 1-1. Output of `java -version`

```
rosencrantz:~ sdavis$ java -version
java version "1.4.2_07"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_07-215)
Java HotSpot(TM) Client VM (build 1.4.2-50, mixed mode)
rosencrantz:~ sdavis$
```

1.4.2. Installing Ant

We use Ant 1.6.5 to compile, package, and deploy the examples in this book. You can download it from <http://ant.apache.org>.

To install Ant, simply unzip the downloaded file to the directory of your choice. Next, create an environment variable called *ANT_HOME* that points to the Ant installation directory. Finally, add *\$ANT_HOME/bin* to the system path so you can run Ant from the command line.

To verify your Ant installation, type `ant -version` at a command prompt. You should see [Example 1-2](#).

Example 1-2. Output of `ant -version`

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

1.5. Installing JBoss

Now that we have all prerequisites in place, we can get to the reason why you are here: installing and running JBoss.

Download the JBoss Application Server Version 4.0.2 from <http://www.jboss.org>. Since it is written in Java, the same installation files will work on Windows, Linux, Unix, or Mac OS X. Any platform that has a JVM can run JBoss.

To install JBoss, simply unzip the downloaded file to the directory of your choice. Next, create an environment variable called *JBOSS_HOME* that points to the JBoss installation directory. Finally, add *\$JBOSS_HOME/bin* to the system path so you can run JBoss from the command line.

To verify your JBoss installation, type *run* at a command prompt (run.bat for Windows users, *run.sh* for Linux/Unix/Mac users). You should see something like this in your terminal window:

```
rosencrantz:/Library/jboss/bin sdavis$ ./run.sh
=====
=====
JBoss Bootstrap Environment

JBOSS_HOME: /Library/jboss

JAVA: /System/Library/Frameworks/JavaVM.framework/home/bin/java

JAVA_OPTS: -server -Xms128m -Xmx128m -Dprogram.name=run.sh

CLASSPATH: /Library/jboss/bin/run.jar:/System/Library/Frameworks/
JavaVM.framework/home/lib/tools.jar

=====
=====

22:14:03,159 INFO [Server] Starting JBoss (MX MicroKernel)...
22:14:03,177 INFO [Server] Release ID: JBoss [Zion] 4.0.2
(build: CVSTag=JBoss_4_0_2 date=200505022023)
22:14:03,181 INFO [Server] Home Dir: /Library/jboss-4.0.2

[many lines deleted for clarity...]

22:14:55,890 INFO [Http11Protocol] Starting Coyote
HTTP/1.1 on http-0.0.0-8080
22:14:56,396 INFO [ChannelSocket] JK: ajp13 listening on /0.0.0.0:8009
22:14:56,519 INFO [JkMain] Jk running ID=0 time=0/240 config=null
22:14:56,530 INFO [Server] JBoss (MX MicroKernel)
[4.0.2 (build: CVSTag=JBoss_4_0_2 date=200505022023)]
Started in 53s:238ms
```

If you don't see any exceptions scroll by, JBoss is up and running when you see the final line: Started in xx ms. To stop JBoss, press *ctrl+C*.

Now that we're sure that everything runs, let's explore JBoss a bit more closely.

1.5.1. Touring the JBoss Directory Structure

JBoss has directory structure that resembles most open source projects (as shown in [Figure 1-1](#)). We're going to briefly point out what each directory holds, and then quickly move along. You'll rarely need to make changes to them. JBoss is configured and ready to run out of the box. You'll spend most of your time messing around with the *server/* directory. This is where you deploy your application.

[Figure 1-1](#) shows a brief overview of each directory:

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

1.6. Deploying Applications to JBoss

J2EE applications are generally bundled up as Enterprise Archives (EARs) or Web Archives (WARs). JBoss services can be bundled up as Service Archives (SARs). While each application technically is nothing more than a simple Java Archive (JAR), they have special internal directory structures and configuration files that must be present for the sake of the application server. (We will discuss EARs and WARs in greater detail later in the book.)

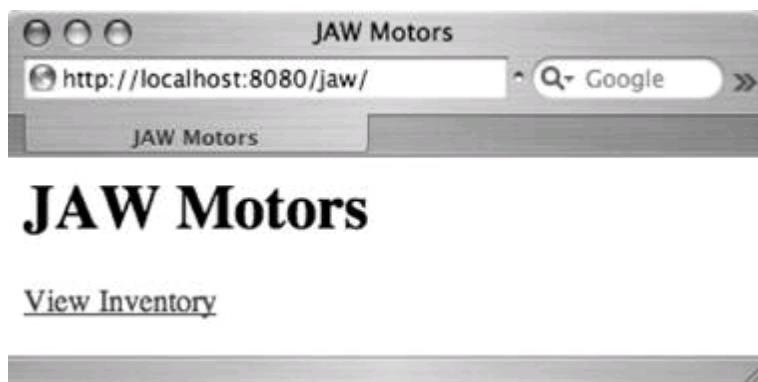
Knowing when to use these different file types and where to place them can be confusing. Here are some basic principles:

- *\$JBoss_HOME/lib* is for the application server's dependent libraries. These file types should always be packaged as JARs. You should never put your own JARs in this directory.
- *\$JBoss_HOME/server/[server configuration]/lib* is for the Server Configuration's dependent libraries. These, too, should always be JARs. You may add an occasional database driver JAR to this directory.
- *\$JBoss_HOME/server/[server configuration]/deploy* is for SARs, WARs, and EARs. Plain old JARs will be ignored if placed here directly, although all three types of files may themselves contain JARs.

If you haven't done so already, go to <http://www.jbossatwork.com> and download the code examples. Once you've unzipped the downloaded file, copy *jaw.war* from the *ch01/01a-test* directory to *\$JBoss_HOME/server/default/deploy*. In the JBoss console window, you should see the deployed test application.

To verify that the application was deployed correctly, open a web browser and go to <http://localhost:8080/jaw> (see [Figure 1-2](#)).

Figure 1-2. JAW Motors web page



Note that our WAR is treated no differently than an MBean. If you move *jaw.war* to the *undeploy/* directory, JBoss will dynamically unload it, as it did with Hypersonic earlier.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

1.7. Looking Ahead...

Now that you've had a chance to see JBoss at work, let's begin writing our web application. The next chapter walks you through the basics of building and deploying a WAR.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

Chapter 2. Web Applications

In this chapter, we'll begin writing the JAW Motors application. First, we'll explain some basic architectural principles like the MVC pattern of web development. Then we'll explore the building blocks of the web tier (JSP, JSTL, CSS, and servlets). Finally, we'll use ANT to create WAR files and deploy them to JBoss.

Our goal is not to teach you these technologies and techniques from scratchthey should be familiar to most web developers already. This chapter is meant to be a quick overview and demonstration of the technology working in JBoss. For deeper look into web tier software and development best practices, look at *Head First Servlets & JSP*, by Bryan Basham, Kathy Sierra, and Bert Bates (O'Reilly).

◀ PREV

NEXT ▶



[◀ PREV](#)

[NEXT ▶](#)

2.1. The Servlet Container

The servlet container is the core J2EE technology that powers the web tier. It is considered a container because JSPs and servlets cannot run as standalone applications—they must implement special interfaces and run "inside" the container. The container listens for HTTP requests on a given port, hands the incoming data to your custom components, and uses the resulting output to create a well-formed HTML document to return to the web browser as the response.

The JBoss developers wisely chose not to create their own servlet container to meet this need. A number of excellent ones already are out there. Rather than recreating the wheel, JBoss allows you to integrate the servlet container of your choice.

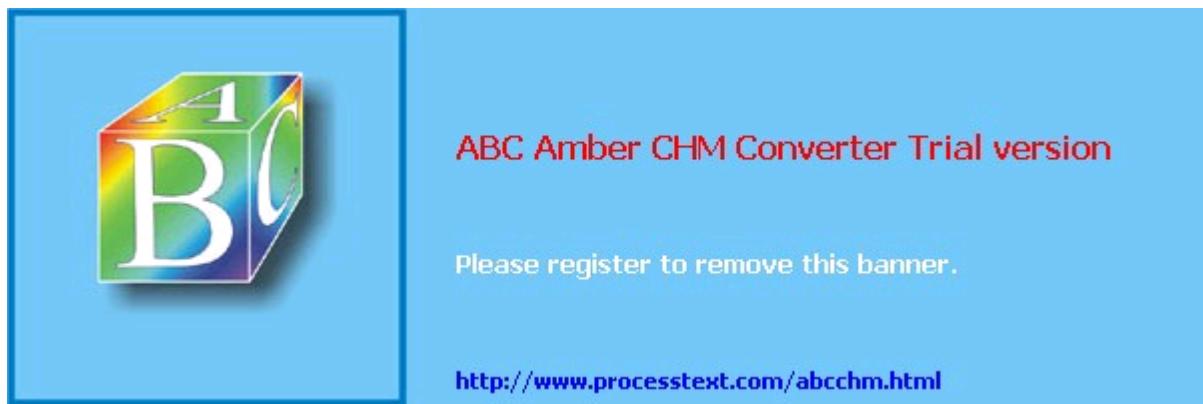
Tomcat 5.5.9 is the default servlet container included with JBoss 4.0.2. It is deployed as a SAR in `$JBOSS_HOME/server/default/deploy/jbossweb-tomcat55.sar`.

If you are comfortable working with Tomcat, you should have no trouble working in JBoss. All the usual configuration files and JARs are in the Tomcat directory. For example, edit `server.xml` to change the port Tomcat listens on. (8080 is the default.) To change the default session timeout from 30 minutes, edit `conf/web.xml`. (For more information about installing and configuring Tomcat, see *Tomcat: The Definitive Guide* by Jason Brittain and Ian F. Darwin (O'Reilly).)

Thanks to the modular design of JBoss, swapping out Tomcat for another servlet container is easy. Jetty is another option that is fast, mature, and open source. (As a matter of fact, it was the default container included with JBoss 3.x.) Go to <http://jetty.mortbay.org/jetty/download.html> to download a pre-built SAR, ready to drop in and run.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

2.2. Three-Tier Applications

Before we dive into writing our code, let's take a moment to talk about some architectural principles. This discussion will set the stage for this chapter and the rest of the book.

The parts of a J2EE application fall into three basic tiers:

- The presentation tier is the collection of J2EE components that comprise the UI. For web applications, this tier includes components like Servlets and JSPs.
- The business tier is so named because it is where your business logic lives. The technologies that live here include EJBs and MDBs.
- The persistence tier is where your data is stored for the long term. It generally involves a relational database, but isn't limited to that. You could store your data as XML, serialized JavaBeans, or even plain old ASCII text files.

In a well-engineered application, components in each tier are "highly cohesive" and "loosely coupled". By highly cohesive, we mean that each component should do only one thing. By loosely coupled, we mean that components across the tiers shouldn't depend on one another unnecessarily.

For example, a `getCars()` method call should return only data. The data shouldn't be preformatted into HTML. If it is, your data tier knows too much about the presentation tier. The two tiers are highly coupled.

An unfortunate side effect of this coupling is that deciding to add a new presentation technology later (like a Swing client) will force you to rewrite a portion of your persistence tier as well. If instead your persistence tier returns data and lets the presentation tier concern itself with the formatting, then you can reuse your persistence tier across multiple UIs. Loosely coupled tiers encourage reuse.

You also could argue that the persistence tier fails the highly cohesive test. It is concerned with too many things—data and presentation. A clean separation of concerns is what we are hoping to accomplish.

This notion of tiers guided both the JAW Motors application architecture and the layout of this book. The book starts with the presentation tier in the first few chapters, moves through the persistence tier in the middle chapters and tackles business tier issues in the latter chapters.

◀ PREV

NEXT ▶



2.3. Exploring the Presentation Tier

So, here we are at the presentation tier . The same principles of high cohesiveness and loose coupling are as applicable within this tier as they are across tiers. In the presentation tier, it is known as the Model / View / Controller (MVC) pattern.

- The view is the actual screen. It is implemented using JSPs, CSS, and taglibs like the JSTL. It is concerned only about formatting the data in a pleasing way.
- The model is the data. In a "View Cars" screen, the model is a Plain Old Java Object (POJO). In our case, it is a simple JavaBean named CarBean .
- The controller mediates communication between the View and the Model. If a "viewCarList" request comes in, it retrieves the CarBeans from the persistence layer and hands them to the view. In a web application, the controller generally is a servlet.

Why Didn't You Use Framework X?

Perhaps the only thing more contentious among programmers than the argument over which text editor to use is which web framework to use. There are over 30 different Java-based web frameworks out there to choose from.

The dirty little secret of choosing a framework is that they are more similar than they are different. They all follow the same basic MVC pattern. So whether you choose to use Struts, JSF, Webwork, Tapestry, or any of the others, you are doing essentially the same thing using a MVC framework.

While the frameworks all have similar philosophies, they differ in terms of implementation. Rather than choosing one and risk having you get lost in the details of the specific framework, we took the coward's way out and rolled our own MVC. We felt that this would allow us to clearly demonstrate how each J2EE technology interacts, and hopefully avoid the wrath of the jilted framework aficionado.

On most projects, we generally go with the framework the team has the most experience with, but you shouldn't feel obligated to choose any of them. We'll use a framework if the added functionality merits the additional overhead, but in the spirit of agile development we'll often use the "home grown" framework presented here for quick and dirty prototypes. By adhering to the basic principles of the MVC pattern, moving from our homegrown framework to a "real" one later in the development cycle is reasonably painless.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

2.4. Building the View Cars Page

Let's begin by implementing the View Cars page. As you can imagine, the first thing most people do when they visit a car dealership is walk around the lot to see which cars the dealer has in stock. In a similar vein, we'll create a view that allows visitors to "walk the lot" via the web.

This view also gives us an excuse to explore the structure of a WAR file. We'll deploy the WAR to JBoss and see it run in container. Once we have that much under our belt, we'll swing back around and do a quick refactoring to implement the model and controller.

2.4.1. Iteration 1: HTML

JSPs are plain HTML files with some templating capabilities that enable us to plug in data dynamically at runtime. Here is an example of our page using static HTML. (We'll add the dynamic content in just a moment.) [Example 2-1](#) should give you a basic feel for what the webpage will look like, and [Figure 2-1](#) shows the result in a web browser.

Example 2-1. carList.html

```
<html>
<body>
  <table border="1">
    <tr>
      <th bgcolor="cccccc" align="left">Make</th>
      <th bgcolor="cccccc" align="left">Model</th>
      <th bgcolor="cccccc" align="right">Model Year</th>
    </tr>

    <tr>
      <td align="left">Toyota</td>
      <td align="left">Camry</td>
      <td align="right">2005</td>
    </tr>

    <tr>
      <td align="left">Toyota</td>
      <td align="left">Corolla</td>
      <td align="right">1999</td>
    </tr>

    <tr>
      <td align="left">Ford</td>
      <td align="left">Explorer</td>
      <td align="right">2005</td>
    </tr>
  </table>
</body>
</html>
```

Figure 2-1. carList.html rendered in a web browser

Make	Model	Model Year
Toyota	Camry	2005
Toyota	Corolla	1999
Ford	Explorer	2005

2.4.2. Iteration 2: JSP and JSTL

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

2.5. Adding a Model and Controller

Now we have a working skeleton. Let's add the model and controller to complete the MVC framework.

2.5.1. The Model

The model is a Plain Old Java Object (POJO). You've built classes like this a thousand times before. `CarBean` is nothing more than a class with three member variables and the associated accessors and mutators.

As you saw in the previous JSP example, using string arrays for data storage doesn't yield very expressive source code. We use an object-oriented programming language to build objects that have a deeper semantic meaning than a pile of primitive data types. In [Example 2-9](#), you can see that `CarBean` is nothing more than a class with three member variables and the associated accessors and mutators. `Car.make`, `Car.model`, and `Car.modelYear` mean something to us, as opposed to the use of `string [0]`, `string [1]`, and `string [2]`.

Example 2-9. CarBean.java

```
package com.jbossatwork;

public class CarBean
{
    private String make;
    private String model;
    private String modelYear;

    public CarBean(String make, String model, String modelYear)
    {
        this.make = make;
        this.model = model;
        this.modelYear = modelYear;
    }

    public String getMake( )
    {
        return make;
    }

    public void setMake(String make)
    {
        this.make = make;
    }

    public String getModel( )
    {
        return model;
    }

    public void setModel(String model)
    {
        this.model = model;
    }

    public String getModelYear( )
    {
        return modelYear;
    }

    public void setModelYear(String modelYear)
    {
        this.modelYear = modelYear;
    }
}
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

2.6. Looking Ahead...

As this application matures, the various pieces get pushed into different tiers. Instead of the servlet creating the `CarBeans` out of thin air, it will pass the request to an EJB. The EJB will eventually get the information out of a database. But for now we completed the first step toward a fully realized J2EE application.

If you are familiar with agile development methodologies, you know that having one working application in the hand is worth two that are still stuck in development. Even though the UI is just the tip of the iceberg in terms of the entire application, having this much in place is a huge milestone.

Nothing makes an application seem more real than a working set of screens, even if (as is the case here) it has nothing of substance behind the facade. It allows the user to see the program far more clearly and persuasively than index cards, white board drawings, or cocktail napkins. From the developer's perspective, it gives a clear idea of what blanks still need to be filled in (persistence, business logic, etc.).

Perhaps most importantly, it allows the UI to have the longest usage cycle. By the time the application is fully implemented, you and the users will have spent enough time working with the UI that all the usability and look-and-feel issues will have been long since been hammered out. As one of our customers said, "Because I was working with the screens from the very beginning, by the time the final application was delivered, I felt like I was already an expert."

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

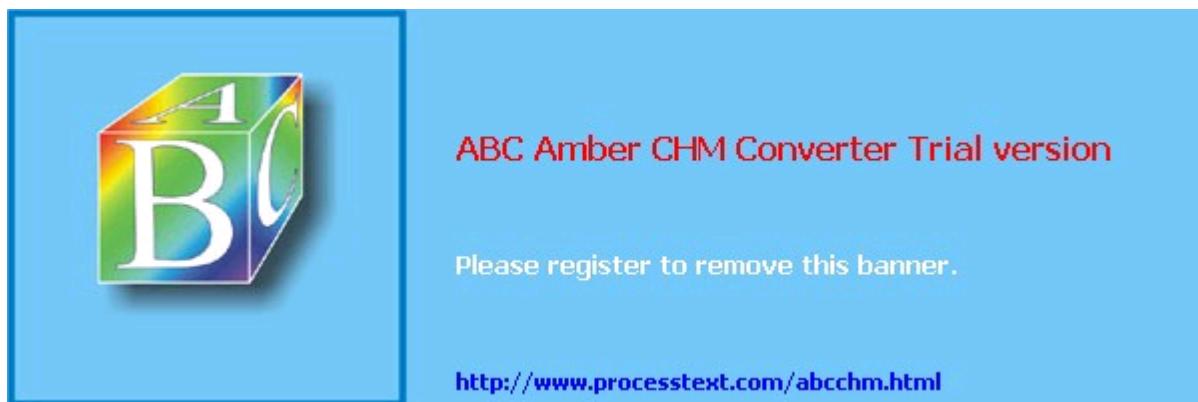
Chapter 3. Building and Deploying an EAR

In the last chapter we introduced you to web applications, but the web tier is just one part of the J2EE spectrum. In this chapter, we'll expand the JAW application from a simple WAR file into a full-fledged EAR.

We'll explore the different parts of an EAR file. We'll build a Common JAR containing classes that can be shared across all tiers of the application. Finally, we'll play with various Ant and XDoclet tasks to create our EAR and dynamically generate the deployment descriptors JBoss needs.

◀ PREV

NEXT ▶



 PREV

NEXT 

3.1. WARs Versus EARs

The WAR file is a convenient way to bundle up all pieces of a web application. All servlet containers know how to deploy a WAR file—they expand the bundle, look for the *WEB-INF* directory, and read the *web.xml* found there for further deployment instructions.

The EAR file provides the same type of functionality for a full-fledged J2EE application. JBoss expands the EAR, finds the required deployment descriptors, and proceeds from there.

An EAR is like a carton of eggs—it keeps everything organized. While the carton doesn't add any direct value to your omelet, it makes getting the eggs home from the store so easy that you wouldn't think about transporting eggs any other way.

Each egg in your EAR carton is a specific piece of the J2EE puzzle. These eggs (or JARs) come in three basic varieties called "modules":

Web module

A WAR file containing presentation tier components

EJB module

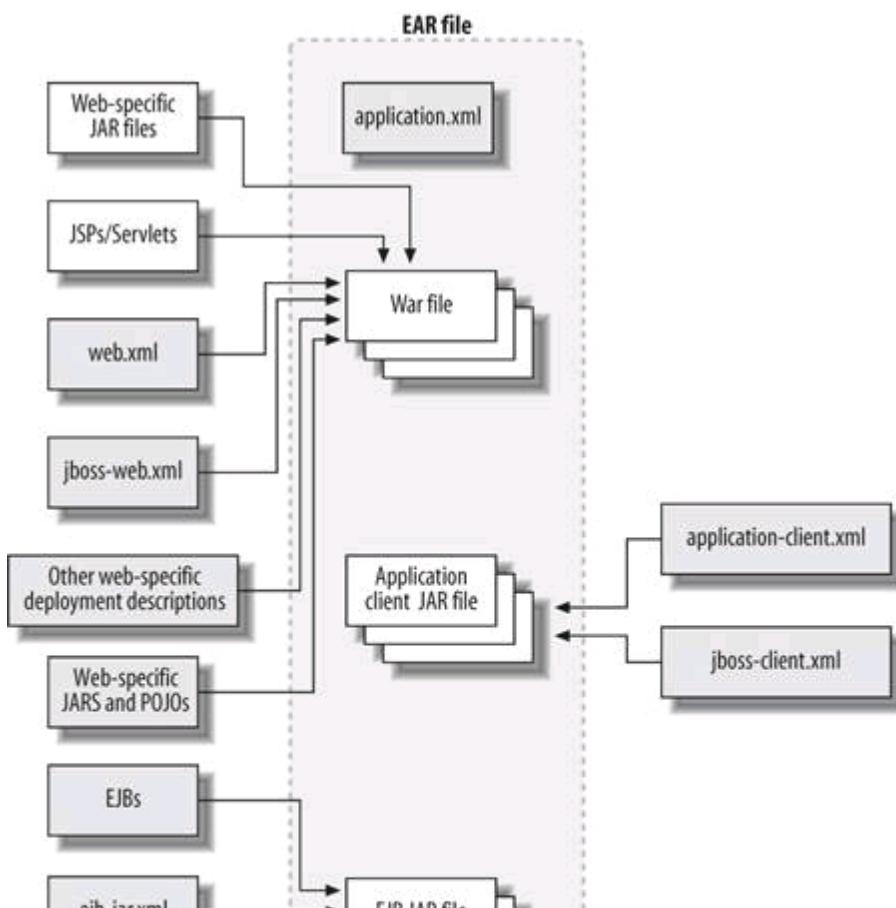
An EJB JAR file containing the middle-tier components (EJBs, MDBs, etc.)

Java module

A regular JAR file containing classes and libraries that are shared across the entire application. An application client JAR and a common JAR are two examples of Java modules.

An EAR can contain at least one of any of these modules. By the same token, any of them can be safely omitted if they aren't needed. [Figure 3-1](#) shows the structure of an EAR file.

Figure 3-1. EAR file structure



◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

3.2. Application.xml

Just as a WAR file contains a *web.xml* deployment descriptor, an EAR file contains a file named *application.xml*. It is essentially a packing list, telling the J2EE server exactly what files the EAR contains and where you can find the files relative to the root of the EAR. The EAR file's *META-INF* directory stores *application.xml*.

[Example 3-1](#) shows the JAW Motors *application.xml* file.

Example 3-1. application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/application_1_4.xsd"
    version="1.4">
<display-name>JBossAtWorkEAR</display-name>

<module>
    <web>
        <web-uri>webapp.war</web-uri>
        <context-root>jaw</context-root>
    </web>
</module>

<module>
    <java>common.jar</java>
</module>

</application>
```

The elements in *application.xml* should be pretty self-explanatory. We are telling the application server the name of each JAR and what function it serves.

Notice that Web modules allow you to specify one other value the *<context-root>*. Recall from the previous chapter that the context root is your web site's URL. If you deploy a simple WAR file, the name of the WAR will be used as the URL. When your WAR file is deployed inside an EAR, this element allows you to override the physical name of the WAR and use whatever URL you'd like.

Although not shown in this example, *<security-role>* is another important element in *application.xml*. The *<security-role>* element describes (what else?) the security roles used throughout a J2EE application for both web and EJB components. Defining security roles in *application.xml* provides a single place to set up J2EE declarative security without duplicating it in *web.xml* and *ejb-jar.xml*. The Security chapter describes *<security-role>* in greater detail.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

3.3. Common JAR

In the previous chapter, we created a `CarBean` POJO to hold the various attributes of a car. We stored it in the WAR file because, well, you didn't have any other choice at that time. We should now reconsider the storage location for the `CarBean` to maximize its reuse.

By the end of this book, we will pull cars out of a database in the persistence tier, and hand them to objects in the business tier, which ultimately will pass them up to the presentation tier. An object that is shared across tiers is a perfect candidate for the Common JAR .

In addition to custom domain objects, the Common JAR is a great location to store common libraries such as Log4J or JDOM. While both WARs and EJB JARs have `lib` directories, they are best used for tier-specific libraries. For example, the JSTL JARs belong in the WARthey have no other purpose than to support the creation of web pages. On the other hand, logging is something that happens throughout the codebaseit really belongs in a common JAR.

Let's factor our `CarBean` out of the WAR and into the Common JAR. In addition to moving directories, we're going to rename it to better describe its purpose in the application.

The suffix "Bean" is a bit overloaded: it includes JavaBeans, Enterprise Java Beans, Session Beans, Message-Driven Beans, JMX MBeans, and the list goes on. The design pattern that best describes the `CarBean`'s function is a Data Transfer Object (DTO), so when we move the bean, we'll also rename it `CarDTO`. The source code will remain the same, but the name will give us a better idea about the true purpose of the class.

3.3.1. Exploring the New Directory Structure

The previous chapter included only the `webapp` directory. If you change to the `ch03/03a-ear` directory, you'll see that we've expanded to a `webapp` directory and a `common` directory.

We also expanded from one to three `build.xml` files. Each subdirectory has its own `build.xml`, and the master build file lives in the top-level directory. The goal is to keep each portion of the application as autonomous as possible. Granted, most of the application will depend on the common sub-project , but by providing individual Ant scripts you have the opportunity to build each portion of the project separately.

3.3.1.1. The common sub-project

Take a moment to explore the `common` sub-project. It contains a single class`CarDTO`. We have created a new package structure to store all DTOs`com.jbossatwork.dto`.

You can build this sub-project by typing `ant` in the common directory. It compiles the `CarDTO` class and bundles it up into a JAR file. After you've built the sub-project, change to the `build/distribution` directory and type `jar tvf common.jar` to verify that the `CarDTO` class is indeed stored in `common.jar`.

3.3.1.2. The webapp sub-project

The only change to the webapp sub-project from the previous chapter is the removal of the `CarDTO` class. To accommodate this change, we now must import the `com.jbossatwork.dto` package at the top of `ControllerServlet`.

We also have to change our `build.xml` script to include the `common.jar` in our classpath. Notice that the definition of `common.jar.dir` uses a relative path to step up one level from the basedir of the `webapp` sub-project and down into the `common` sub-project's output directory in [Example 3-2](#).

Example 3-2. webapp build.xml

```
<property name="lib.dir" value="lib"/>
<property name="compile.lib.dir" value="compile-lib"/>
<property name="common.jar.dir" value="../common/build/distribution"/>

<path id="compile.classpath">
    <fileset dir="${compile.lib.dir}">
        <include name="*.jar" />
    </fileset>
    <zipfileset dir="${common.jar.dir}" files="common.jar" />
</path>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

3.4. Deploying the EAR

Let's make sure that the WAR files from the previous chapters don't end up conflicting with our EAR file. Delete `jaw.war` from the `deploy` directory before moving on. Now we're ready to drop in our newly created EAR file.

Deploying an EAR by hand is no different than deploying a WAR by hand. Copy `jaw.ear` to `$JBoss_HOME/server/default/deploy`. Your JBoss console should let you know that it was deployed successfully, as in [Example 3-6](#).

Example 3-6. JBoss console output showing a successful EAR deployment

```
22:37:55,659 INFO [EARDeployer] Init J2EE application:  
    file:/Library/jboss-4.0.1/server/default/deploy/jaw.ear  
22:37:55,853 INFO [TomcatDeployer] deploy, ctxPath=/jaw,  
    warUrl=file:/Library/jboss-4.0.1/server/default/tmp/deploy/  
    tmp25111jaw-ear.ear-contents/webapp.war/  
22:37:56,159 INFO [EARDeployer] Started J2EE application:  
    file:/Library/jboss-4.0.1/server/default/deploy/jaw-ear.ear
```

Automated Deployments Using Ant

Ant allows you to do far more than simply compile and bundle up your Java application. It has tasks that let you create and delete directories and copy files to a local subdirectory or a remote server by using ftp or scp. Using tasks like `exec`, `rexec`, or `sshexec`, you can even remotely start and stop JBoss.

It's not hard to imagine completely automating the deployment process with an Ant script. But just because you *can* do something doesn't always mean you *should*.

Automating deployment to a test server certainly will help speed up your development iterations. But deployments to a production server should be taken a bit more seriously. Upgrading a production application is something that should be done deliberately, and in our opinion, should be done by hand.

If you provide an Ant task to deploy your application to a production server, you almost certainly guarantee that you will invoke it accidentally at the most inopportune time.

Since we point at a test server for this book, we've provided a convenient couple of Ant targets to deploy your EAR. To do a hot deploy, make sure that you have the `$JBoss_HOME` environment variable set, and then simply type `ant deploy`. The Ant task will copy the EAR file to the correct location for you.

If you want to do a cold deploy, shut down JBoss, type `Ant cleandeploy`, and then start JBoss back up again. The `cleandeploy` target will delete the existing EAR file and several temporary directories. Running `cleandeploy` against a running JBoss instance will cause bad things to happen, so make sure that JBoss is not running before invoking it.

Visit <http://localhost:8080/jaw> to confirm that the application was indeed deployed and still works as expected. Yes, this application doesn't look or behave any differently than the one in the Web chapter. But we added hundreds of lines of new codeisn't that the true measure of a successful J2EE project? (Only kidding....)

In all seriousness, we haven't added any new functionality that the user would notice, but we have set the stage for easy future growth and maximum flexibility. Knowing that the other tiers are coming up soon, these changes will allow you to incorporate the new technology with minimal effort.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

3.5. Adding a DAO

In that spirit, let's add another component that will pay dividends in future flexibility. Right now, your servlet is creating the car list each time a request comes in. This really isn't optimal. Servlets should deal with the mechanics of the HTTP request/response cycle. They shouldn't perform persistence tier tasks.

We aren't quite ready to install a database (that happens in the next chapter), but we can lay the groundwork by creating a Data Access Object (DAO). A DAO is a layer of abstraction it hides the actual persistence specifics behind a common interface.

The DAO we create in this chapter still stores the DTO objects in a simple `ArrayList`. In the next chapter, the DAO will pull car data from a database that uses JDBC. In the chapter after that, it will use Hibernate (an Object/Relational Mapper) to do the same thing. By getting the DAO in place now, however, we'll be able to make these implementation changes without affecting presentation-tier code. Loose coupling and high cohesion comes to the rescue again.

The `CarDAO` provides a `findAll()` method that returns a `List` of `CarDTOs`. The source code in [Example 3-7](#) can be found in the `common` directory in `ch03b-dao`.

Example 3-7. CarDAO.java

```
package com.jbossatwork.dao;

import java.util.*;
import com.jbossatwork.dto.CarDTO;

public class CarDAO
{
    private List carList;

    public CarDAO( )
    {
        carList = new ArrayList( );

        carList.add(new CarDTO("Toyota", "Camry", "2005"));
        carList.add(new CarDTO("Toyota", "Corolla", "1999"));
        carList.add(new CarDTO("Ford", "Explorer", "2005"));
    }

    public List findAll( )
    {
        return carList;
    }
}
```

The corresponding change in the `ControllerServlet` calls the newly created DAO in [Example 3-8](#).

Example 3-8. ControllerServlet.java

```
// perform action
if(VIEW_CAR_LIST_ACTION.equals(actionName))
{
    CarDAO carDAO = new CarDAO( );
    request.setAttribute("carList", carDAO.findAll( ));

    destinationPage = "/carList.jsp";
}
```

Not only does this change simplify the code in the servlet, it *feels* more correct as well. The

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

3.6. Using XDoclet

The last thing we'll do in this chapter is one more tiny bit of automation.

One of the most important parts of the WAR file is the deployment descriptor `web.xml`. It lists each servlet and tells the servlet container how to deploy them. By maintaining this file by hand, you almost certainly guarantee that you will forget to update it when you add new components to your application. By using XDoclet in your build process, we can generate it automatically by using nothing but your source code and some well-placed comments.

XDoclet is a collection of custom Ant tasks that generate code during the build process. Sometimes XDoclet can generate the necessary code or deployment descriptor using just the source code. At other times, you'll need to add custom comments to nudge XDoclet along in the right direction.

We have only one servlet right now `ControllerServlet`. [Example 3-9](#) shows what your `web.xml` must contain to deploy this servlet correctly.

Example 3-9. web.xml

```
<!-- servlet definition -->
<servlet>
    <servlet-name>Controller</servlet-name>
    <servlet-class>com.jbossatwork.ControllerServlet</servlet-class>
</servlet>

<!-- servlet mapping -->
<servlet-mapping>
    <servlet-name>Controller</servlet-name>
    <url-pattern>/controller/*</url-pattern>
</servlet-mapping>
```

Look in `ch03c-xdoclet/src` for the new `ControllerServlet` source code in [Example 3-10](#). Notice the XDoclet comments we've added.

Example 3-10. ControllerServlet.java

```
/**
 * @web.servlet
 *     name="Controller"
 *
 * @web.servlet-mapping
 *     url-pattern="/controller/*"
 */
public class ControllerServlet extends HttpServlet
```

See how they correspond to the `web.xml` elements? Now let's look at `build.xml` to see the newly added XDoclet Ant tasks. We first need to create a couple of new variables, as in [Example 3-11](#).

Example 3-11. Defining XDoclet variables in build.xml

```
<property name="xdoclet.lib.dir" value="${env.XDOCLET_HOME} /lib"/>
<property name="gen.source.dir" value=" ${build.dir}/gensrc"/>
```

`xdoclet.lib.dir` points to the XDoclet jars. If we have a defined `$XDOCLET_HOME` environment variable, `build.xml` should be automatically pointed in the right direction. (Recall that we installed XDoclet in [Chapter 1](#).)

The second variable defines a new location for our dynamically generated code. By keeping our compiled code and generated source code in the same location (`build.dir`), we can easily delete and recreate it each time we run the Ant tasks. It also gently reminds us that we

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

3.7. Looking Ahead...

We've covered a lot of ground in these first few chapters. We now have a single EAR file that encapsulates many moving parts of our J2EE application. We have automated the build process where it is appropriate, and set the stage for future growth.

In the next several chapters, we'll leave the web tier and move on to the persistence tier. We've stored our DTOs in an `ArrayList` for long enough let's tackle saving them in a true database.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

Chapter 4. Databases and JBoss

Up to this point, this book has focused on the web tier. Now let's look at the persistence tier. This is where the application data is stored for the long term for example, between server restarts.

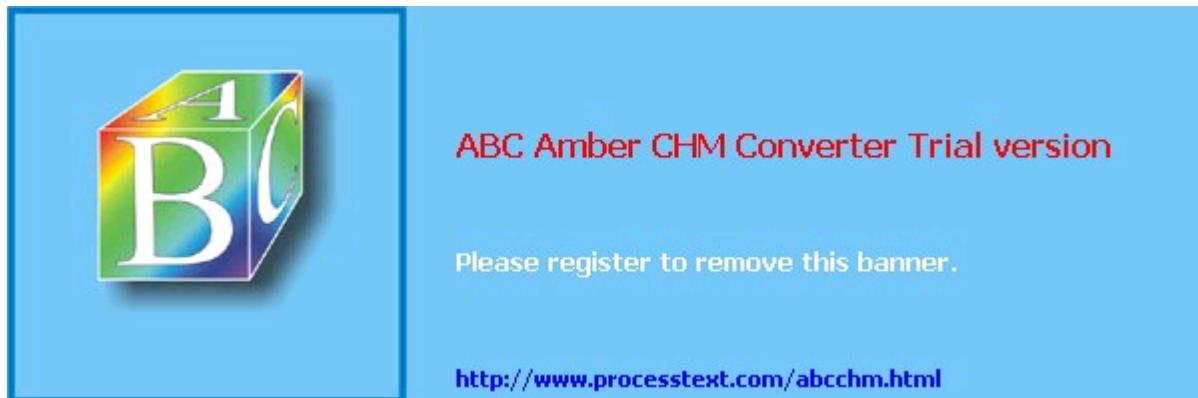
Why use the phrase "persistence tier" instead of simply calling it the "database tier"? We certainly recognize that the probability of information ending up in a database approaches is somewhere close to 100%. J2EE pundits love pointing out that data *could* be stored in any number of manners as flat files, XML, and even web services to remote servers. These types of storage are mentioned as alternatives, but we have yet to work on an application where they completely usurp the trusty database.

Instead, most modern persistence technologies deal with transforming relational database information into Java objects. These Object/Relational Mappers (ORMs) come in many flavors commercial and open source but make the same promise: to free the Java developer from the perils of converting ResultSets to ArrayLists of DTOs.

We continue to use the phrase "persistence tier" to remind us that many supporting services surround the inevitable database.

◀ PREV

NEXT ▶



 PREV

NEXT 

4.1. Persistence Options

You should acknowledge one simple fact up front: if you deal with a relational database, all roads in one form or another lead to JDBC. Whether you write the code yourself or let an ORM write it for you, SQL `INSERTs`, `UPDATEs`, and `DELETEs` are the lingua franca of any database-driven application.

While Sun maintains that JDBC is not an acronym, it looks suspiciously like "Java DataBase Connectivity" to many seasoned programmers. It is the API that allows us to load up database `Drivers`, make `Connections`, and create `Statements` that yield `ResultSets` upon execution.

While nothing is intrinsically wrong with `ResultSets`, OO purists bristle at the thought of dealing with a semi-structured collection of strings and primitives. Java programmers are taught from a tender young age that JavaBeans and DTOs are the one true way to represent business objects. So to get from `ResultSets` to DTOs, we must use hand-coded methods that do the transformation for us, one `car.setName(resultSet.getString("name"))` at a time.

While this isn't terribly difficult, it does get tedious as the number of business objects and database tables grow. Maintaining two separate data schemas, one in Java and the other in SQL, strikes many as a flagrant violation of the DRY principle. The phrase "impedance mismatch" often comes up in JDBC discussions.

One potential way to avoid the problem of marshalling and unmarshalling JavaBeans is to remove the root cause why not just create a database that deals natively with objects? On paper, object-oriented databases (OODBMS) seem to be the ideal solution to this problem. Sadly, OODBMSes have never gained any serious market share.

If you can't change the root data source and relational databases are deeply entrenched in most long-term persistence strategies, your only other option is to come up with an API that manages the impedance mismatch: something that allows you to deal with native JavaBeans, and not only hides the JDBC complexity from you, but ideally entirely creates and manages the infrastructure.

One of the earliest attempts at this was the now infamous Entity Bean offering in the EJB specification. Entity beans came in two basic variations: Bean-Managed Persistence (BMPs) and Container-Managed Persistence (CMPs).

BMPs were really nothing more than a fancy way of saying, "I'm going to keep on doing the JDBC wrangling that I've already been doing." Since the Bean was responsible for its own persistence implementation, many programmers fell back on what they knew best `car.setName(resultSet.getString("name"))`.

CMPs were closer to what we were hoping to achieve—"let me define the business object and then have the container worry about how to persist it." The problem with CMPs ended up being twofold:

- Rather than dealing with a simple POJO, you were forced to create and maintain a complicated variety of interdependent classes and interfaces `Remotes`, `RemoteHomes`, `Locals`, `LocalHomes`, and abstract bean classes.
- The resulting tangle of code was tightly coupled to the container and very intrusive—you were forced to inherit from `EJBObject` and implement specific interfaces rather than following an inheritance tree that more closely modeled your business domain.

While Entity Beans still exist in the EJB specification today, they have largely fallen out of favor in the developer community.

Sun's next attempt at a JavaBean-centric persistence API was Java Data Objects (JDO). The 1.0 specification has been out for several years, but it hasn't captured a lot of mindshare. Some point to a differently but equally complicated API as its main problem. Traditional RDBMS vendors have been slow to support it, although OODBMS vendors have enthusiastically touted it as the Next Big Thing. Regardless, JDO is not an official part of the J2EE specification, so it has gone largely unnoticed by the server-side crowd.

Which leads us to the wild west of independent ORMs. Many solutions both commercial and

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

4.2. JDBC

JDBC has been around nearly as long as Java itself. The JDBC 1.0 API was released with JDK 1.1. This is the `java.sql` package. JDBC 2.0 was released with JDK 1.2. It included both the Core package and what was called the Optional Package (`javax.sql`). The optional package brought with it better enterprise support for database connections, including connection pools and distributed transactions. JDBC 3.0 is the latest release, included with JDK 1.4.

If you've written JDBC code since the good old days, you're probably familiar with using the `DriverManager` to get a database connection, as in [Example 4-1](#).

Example 4-1. Example of the JDBC DriverManager

```
static final String DB_DRIVER_CLASS = "com.mysql.jdbc.Driver";
static final String DB_URL =
    "mysql://localhost:3306/JBossatWorkDB?autoReconnect=true";

Connection connection = null;

try {
    // Load the Driver.
    Class.forName(DB_DRIVER_CLASS).newInstance();

    // Connect to the database.
    connection = DriverManager.getConnection(DB_URL);

} catch (SQLException se) {
    ...
} catch (...) {
    ...
}
```

While this code certainly works, it has several shortcomings:

- Every time you connect and disconnect from the database, you incur the overhead of creating and destroying a physical database connection.
- You have to manage the database transaction yourself.
- You have a local transaction that's concerned only with database activity. What if you deal with other resources such as JMS Destinations (Queues and Topics)? If there's a problem and you need to roll back database updates, there's no automated way to roll back the work done with these other resources.

One of the main benefits of living in an application server is having the server take care of these sorts of plumbing issues. JBoss, like all other J2EE application servers, deals with the issues listed above on your behalf. However, to facilitate this, we need to slightly change the way you obtain your database connections.

Rather than using a `java.sql.DriverManager`, we need to use a `javax.sql.DataSource` to allow JBoss to manage the details in [Example 4-2](#).

Example 4-2. Example of the JDBC DataSource

```
static final String DATA_SOURCE=
    "java:comp/env/jdbc/JBossAtWorkDS";

DataSource dataSource = null;
Connection conn = null;

try {
    // Load the Driver.
    dataSource = ServiceLocator.getDataSource(DATA_SOURCE);
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

4.3. JNDI

Let's take a moment to parse the `DataSource` name `java:comp/env/jdbc/JBossAtWorkDS`, which is a Java Naming and Directory Interface (JNDI) name. JNDI provides access to a variety of back-end resources in a unified way.

JNDI is to Java Enterprise applications what Domain Name Service (DNS) is to Internet applications. Without DNS, you would be forced to memorize and type IP addresses like 192.168.1.100 into your web browser instead of friendly names like <http://www.jbossatwork.com>. In addition to resolving host names to IP addresses, DNS facilitates sending email between domains, load-balancing web servers, and other things. Similarly, JNDI maps high-level names to resources like database connections, JavaMail sessions, and pools of EJB objects.

DNS has a naming convention that makes it easy to figure out the organizational structure of a Fully Qualified Domain Name (FQDN). Domain names are dot-delimited and move from the general to the specific as you read them from right-to-left. "com" is a Top-Level Domain (TLD) reserved for commercial businesses. There are a number of other TLDs, including "edu" for educational institutions, "gov" for government entities, and "org" for non-profit organizations.

The domain name reserved for your business or organization is called a Mid-Level Domain (MLD). [Jbossatwork.com](http://www.jbossatwork.com), [apache.org](http://www.apache.org), and [whitehouse.gov](http://www.whitehouse.gov) are all MLDs. You can create any number of subdomains under a MLD, but the left-most element will always be a HostName like "www" or "mail."

Now looking at a domain name like <http://www.parks.state.co.us> or <http://www.npgc.state.ne.us> for a listing of state parks in Colorado or Nebraska begins to make a little more sense. The country/state/department hierarchy in the domain name mirrors the real-life organizational hierarchy.

JNDI organizes its namespace using a naming convention called Environmental Naming Context (ENC). You are not required to use this naming convention, but it is highly recommended. ENC JNDI names always begin with `java:comp/env`. (Notice that JNDI names are forward slash-delimited instead of dot-delimited and read left-to-right.)

A number of TLD-like top-level names are in the ENC. Each JNDI "TLD" corresponds to a specific resource type, shown in [Table 4-1](#).

Table 4-1. J2EE-style JNDI ENC naming conventions

Resource type	JNDI prefix
Environment Variables	<code>java:comp/env/var</code>
URL	<code>java:comp/env/url</code>
JavaMail Sessions	<code>java:comp/env/mail</code>
JMS Connection Factories and Destinations	<code>java:comp/env/jms</code>
EJB Homes	<code>java:comp/env/ejb</code>
JDBC DataSources	<code>java:comp/env/jdbc</code>

I'm obviously mixing my JNDI and DNS nomenclature, but the JNDI "TLD" for `DataSources` always should be `java:/comp/env/jdbc`. In the example `DataSource` name `java:comp/env/jdbc/JBossAtWorkDS` the "TLD" and "MLD" should be more self-evident now. `JBossAtWorkDS` is the JNDI "MLD."

DNS names protect us from the perils of hardcoded IP addresses. A change of server or ISP (and the corresponding change in IP address) should remain transparent to the casual end

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

4.4. JNDI References in web.xml

In previous chapters, we used the *web.xml* file to describe and deploy servlets. This same file describes and deploys JNDI resources. The new *web.xml* looks like [Example 4-4](#).

Example 4-4. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <servlet>
        <servlet-name>Controller</servlet-name>
        <servlet-class>com.jbossatwork.ControllerServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Controller</servlet-name>
        <url-pattern>/controller/*</url-pattern>
    </servlet-mapping>

    <resource-ref>
        <res-ref-name>jdbc/JBossAtWorkDS</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>

</web-app>
```

Let's examine each new element:

- <res-ref-name> is the JNDI resource name. Notice that you don't have to specify "java:comp/env/" it is assumed, just like "[http://](#)" is commonly left out of web URLs.
- <res-type> in our case is a DataSource. This must be the fully qualified classname.
- <res-auth> can be either Container or Servlet. Since we use JBoss' DataSource pooling, Container is the appropriate choice here.

OK, so here's where it gets interesting. At first glance, it appears that JBoss doesn't adhere to the ENC naming style when it comes to [DataSources](#). Instead of [java:comp/env/jdbc/JBossAtWorkDS](#), its [DataSources](#) are referenced as simply [java:/JBossAtWorkDS](#). So we need a way to map the JBoss name to the ENC name.

The real reason for the mismatch is that JBoss creates a global binding for the [DataSource](#), and we need to create a local reference to it. We mentioned earlier in the chapter that all JNDI references are local to the EAR. Out of courtesy, JBoss doesn't automatically expose global references to us. We need to map the global name to a local name so that we can work with it.

Luckily, a straightforward way to do the cross mapping is available. You can include a JBoss specific deployment descriptor in your WAR named *jboss-web.xml*. [Example 4-5](#) shows what ours should look like.

Example 4-5. jboss-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
    <resource-ref>
        <res-ref-name>jdbc/JBossAtWorkDS</res-ref-name>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

4.5. JBoss DataSource Descriptors

Remember in [Chapter 1](#) we dynamically deployed and undeployed a service? We used the Hypersonic database in the example. You can access any database as an MBean by simply including the appropriate `*-ds.xml` file in the deploy directory.

Hypersonic is completely implemented in Java and ships standard with JBoss. It is great for playing around with JDBC and not having to worry about installing and configuring an external database. We generally rely on a full-fledged external database for production applications, but we'd be lying if we told you that we didn't use Hypersonic *all the time* for rapid testing and prototyping.

Three types of Hypersonic instances include:

- The default Hypersonic configuration, which gives you a local database whose modifications are saved to disk (and therefore survive between JBoss restarts). We can access this configuration only through a `DataSource` it is not accessible to out-of-container clients like Ant or third-party standalone GUIs. It is called an "In-Process Persistent DB".
- As a slight variation, we can configure the "In-Process Persistent DB" to run purely in memory. No files are written to disk, and therefore the database lives only as long as the container is running. This is called an "In-Memory DB."
- If you need to access the database from either a `DataSource` or an external client, you can configure Hypersonic to listen on a TCP port (1701 by default). This is called a "TCP DB."

The Hypersonic deployment descriptor is `$JBOSS_HOME/server/default/deploy/hsqldb-ds.xml`. Examples of deployment descriptors for all major databases (commercial or open source) are at `$JBOSS_HOME/docs/examples/jca`. The J2EE Connector Architecture (JCA) is a standard way for a J2EE container to connect to external datastores. These example files generally are very well commented. Take a moment to browse the `examples/jca` directory and look through some of the deployment descriptors.

We provide two customized Hypersonic database descriptors in the `ch04/sql` directory. `Jaw-ds.xml` strips out all the comments included in the original `hsqldb-ds.xml` file sometimes it can be hard to see the forest for the trees. We also included a version that retains the original comments. You might like to compare this version to the default Hypersonic version to see how we've tweaked it.

Let's step through `jaw-ds.xml` line by line.

```
<datasources>

<local-tx-datasource>
  <jndi-name>JBossAtWorkDS</jndi-name>
```

This is the global/JBoss JNDI name of your `DataSource`. Since this `DataSource` is accessible to all EARs, it only makes sense to bind its name in the global context. (The local ENC name goes with the local EAR in `web.xml`.)

```
<connection-url>jdbc:hsqldb:hsq://localhost:1701</connection-url>
<driver-class>org.hsqldb.jdbcDriver</driver-class>
<user-name>sa</user-name>
<password></password>
```

These values should look familiar to you. They are the standard JDBC parameters that tell you how to connect to the database, which driver to use, and what credentials to supply when connecting.

```
<min-pool-size>5</min-pool-size>
<max-pool-size>20</max-pool-size>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

4.6. JDBC Driver JARs

A `DataSource` is a container-managed resource. The JBoss documentation recommends storing the JAR outside of your EAR and in `$JBoss_HOME/server/default/lib`. (One big reason for this is that JDBC drivers cannot be hot deployed.) For example, the `$JBoss_HOME/server/default/lib` directory is where you'll find `hsqldb.jar` the JDBC driver for the Hypersonic database. As an added benefit, if you store the drivers here, you can share them across multiple EARs. With less duplication, there is less of a chance for mismatched drivers and database versions.

Figure 4-1. The JBoss JMX-Console

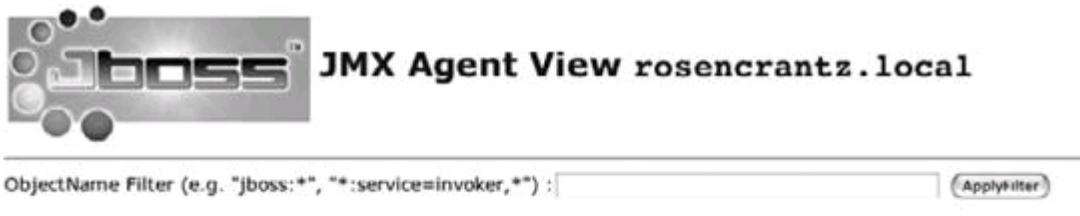


Figure 4-2. MBean configuration

MBean description:

Management Bean.

List of MBean attributes:

Name	Type	Access	Value	Description
State	int	R	3	MBean Attribute.
ShutdownCommand	java.lang.String	RW		MBean Attribute.
User	java.lang.String	RW	sa	MBean Attribute.
Silent	boolean	RW	<input checked="" type="radio"/> True <input type="radio"/> False	MBean Attribute.
Persist	boolean	RW	<input checked="" type="radio"/> True <input type="radio"/> False	MBean Attribute.
Port	int	RW	1701	MBean Attribute.
Password	java.lang.String	RW		MBean Attribute.
StateString	java.lang.String	R	Started	MBean Attribute.
DatabaseManagerClass	java.lang.String	RW	org.hsqldb.util.DatabaseM	MBean Attribute.
Database	java.lang.String	RW	jawdb	MBean Attribute.
No_system_exit	boolean	RW	<input checked="" type="radio"/> True <input type="radio"/> False	MBean Attribute.
Trace	boolean	RW	<input type="radio"/> True <input checked="" type="radio"/> False	MBean Attribute.
InProcessMode	boolean	RW	<input type="radio"/> True <input checked="" type="radio"/> False	MBean Attribute.
Name	java.lang.String	R	HypersonicDatabase	MBean Attribute.
DatabasePath	java.lang.String	R	/Library/jboss-4.0.1/server/default/data/hypersonic/jawdb	MBean Attribute.

Apply Changes

Of course, if you are not going to be hot deploying your EARs you can include your JDBC drivers in your EAR. This gives you the added benefit of allowing each EAR to use potentially different or conflicting versions of the same JDBC driver.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

4.7. Database Checklist

OK, so here's the checklist of things we've accomplished so far:

- Stored the JDBC driver in `$JBOSS_HOME/server/default/lib (hsqldb.jar)`
- Configured the database deployment descriptor in `$JBOSS_HOME/server/default/deploy (hsqldb-ds.xml)` Among other things, this is where we set up the JBoss JNDI name (`java:/JBossAtWorkDS`).
- We created a global JNDI reference to the DataSource in `jboss-web.xml`. This name matches the name in the database deployment descriptor. We also provided a setting that maps the global JNDI name to a local JNDI name using ENC-style naming (`java:comp/env/jdbc/JBossAtWorkDS`).
- We created a local JNDI reference to the DataStore in `web.xml`..
- We created a ServiceLocator class that encapsulates our JNDI lookup and returns the DataSource.

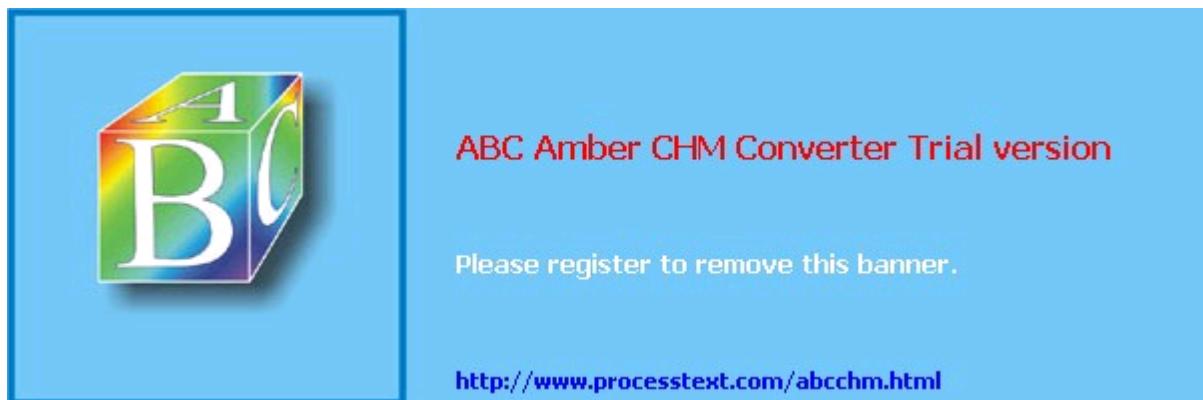
Because of the way we've set things up, switching databases at this point is relatively easy. For example, if you'd prefer to work against an instance of MySQL, we only need to copy the JDBC drivers to the `$JBOSS_HOME/server/default/lib` directory and copy a new database deployment descriptor into the `deploy` directory. If you use the same JNDI name that we already used, your job is done all the code upstream will be configured and ready to hit the new database.

We've said it many times before, but it's worth saying again: Hypersonic is a great database for our immediate purposes because it doesn't require configuring an external resource. However, in a production environment, we'd most likely use a more robust database.

We are now ready to create a `Car` table and insert some sample data.

◀ PREV

NEXT ▶



 PREV

NEXT 

4.8. Accessing the Database Using Ant

Now that we've created the `JBossAtWorkDB` database instance, we need to create and populate the `Car` table. Ant has a `<sql>` task that is ideal for this sort of thing. Keeping these commands in a script allows you to rebuild your database easily and often during the development phase.

The same rules for scripting the deployment of your EAR to a production server apply here as well: Just Say NO! If you create a script that points to a production database, you are only asking for it to be run inadvertently with disastrous results. With great power comes great responsibilityuse it wisely.

That said, let's look at the `build.xml` file in the new `SQL` subproject, shown in [Example 4-9](#). This project doesn't contain any compiled code. It is just a convenient storage location for these SQL scripts.

Example 4-9. SQL subproject build.xml

```
<?xml version="1.0"?>

<project name="sql" default="init" basedir=".">
    <!-- Initialization variables -->
    <property name="database.driver.dir"
              value="${env.JBOSS_HOME}/server/default/lib/" />
    <property name="database.driver.jar" value="hsqldb.jar"/>

    <path id="sql.classpath">
        <fileset dir="${database.driver.dir}">
            <include name="${database.driver.jar}" />
        </fileset>
    </path>

    <!-- ===== -->
    <target name="init"
          description="Creates test data in the database.">

        <sql driver="org.hsqldb.jdbcDriver"
             url="jdbc:hsqldb:hsq://localhost:1701"
             userid="sa"
             password=""
             print="yes"
             classpathref="sql.classpath">

            DROP TABLE IF EXISTS CAR;

            CREATE TABLE CAR (
                ID BIGINT identity,
                MAKE VARCHAR(50),
                MODEL VARCHAR(50),
                MODEL_YEAR VARCHAR(50)
            );

            INSERT INTO CAR (ID, MAKE, MODEL, MODEL_YEAR)
            VALUES (99, 'Toyota', 'Camry', '2005');

            INSERT INTO CAR (ID, MAKE, MODEL, MODEL_YEAR)
            VALUES (100, 'Toyota', 'Corolla', '1999');

            INSERT INTO CAR (ID, MAKE, MODEL, MODEL_YEAR)
            VALUES (101, 'Ford', 'Explorer', '2005');

            SELECT * FROM CAR;
        </sql>
    </target>
</project>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

4.9. Creating JDBC CarDAO

Our first `CarDAO` was fine to get the project kick-started, but `ArrayLists` aren't the best long-term persistence strategy. Let's create a second DAO that takes advantage of the infrastructure we've just put in place.

Since two classes provide different implementations of the same functionality, we should create a common Interface. In addition to making it trivial to switch back and forth between the two concrete implementations, it will also pave the way for us to add a third DAO implementation for Hibernate in the next chapter.

In the `ch04/common` source tree, notice that we renamed our old DAO class to `InMemoryCarDAO`. We didn't touch any of the methods, just the name of the class and the corresponding constructor names, as in [Example 4-11](#).

Example 4-11. CarDAO.java

```
package com.jbossatwork.dao;

import java.util.*;
import com.jbossatwork.dto.CarDTO;

public class InMemoryCarDAO implements CarDAO
{
    private List carList;

    public InMemoryCarDAO( )
    {
        carList = new ArrayList( );

        carList.add(new CarDTO("Toyota", "Camry", "2005"));
        carList.add(new CarDTO("Toyota", "Corolla", "1999"));
        carList.add(new CarDTO("Ford", "Explorer", "2005"));
    }

    public List findAll( )
    {
        return carList;
    }
}
```

The `CarDAO` Interface simply defines the method signature for `findAll()`:

```
package com.jbossatwork.dao;

import java.util.*;

public interface CarDAO
{
    public List findAll( );
}
```

The new `JDBCCarDAO` uses the new `DataSource` and `ServiceLocator` class to build the `ArrayList` of `CarDTOs` in [Example 4-12](#).

Example 4-12. JDBCCarDAO.java

```
package com.jbossatwork.dao;

import java.util.*;
import java.sql.*;
import javax.sql.*;
import com.jbossatwork.dto.CarDTO;
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

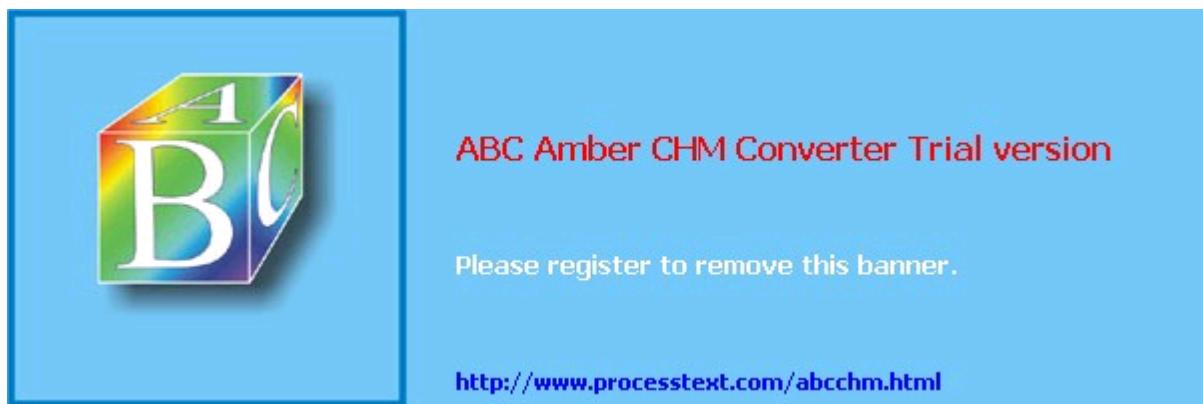
4.10. Looking Ahead...

Once again, we added hundreds of lines of new code with no visible difference to the application. Hopefully you can appreciate what is transparent to the end user by layering your application correctly, you can make massive changes to the persistence tier while leaving your presentation tier virtually untouched.

We have one more iteration of the `CarList` example to get through before we move on to more exciting stuff. In the next chapter, we'll create a Hibernate DAO that drastically simplifies the object-relational mapping that we have to do by hand in the JDBC DAO. After that, we'll start adding some new functionality to the application, such as logging in and buying cars.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

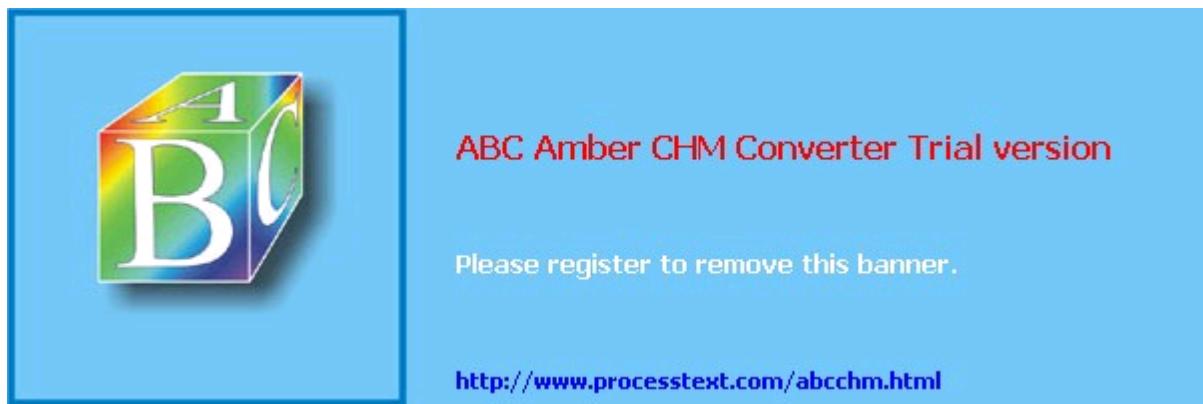
Chapter 5. Hibernate and JBoss

Hibernate is not an official part of the J2EE specification. It is an Object/Relational Mapper that hides the complexity of marshalling and unmarshalling JavaBeans and ResultSets.

Hibernate is unique because it can run either in a J2EE container such as JBoss, or as a standalone service. JBoss 4.0.2 ships standard with Hibernate 3.0.2. This chapter focuses on integrating the two to streamline the persistence tier of your application. (For a more detailed look at installing and running Hibernate, see *Hibernate: A Developer's Notebook* by James Elliott (O'Reilly).)

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

5.1. The Pros and Cons of ORMs

"Object/Relational mapping is the Vietnam of computer science..."Ted Neward (author, Effective Enterprise Java (Addison-Wesley))

If you've read Neward's work or heard him speak, you know that he is a smart, controversial, and very passionate technologist. We couldn't think of a better sentiment to begin our chapter on ORMs.

His point is that the United States started in Vietnam by sending over a few advisors. Then we began to ship in limited numbers of ground troops. Before too long, things degenerated into a full-blown, messy, unpopular war where we were heavily committed and had a difficult time extricating ourselves.

Working with ORMs (in his mind) is really no different. You start out with simple objects that map neatly to single rows. Then you get brave and begin using composition (classes within classes). Hibernate is smart enough to handle that, but at some point you are going to come up with either a complicated object model or a highly normalized relational database that doesn't map nicely through Hibernate.

Our recommendation is to tread lightly. ORMs like Hibernate are relatively new in the Java world. The related specifications are moving quickly and have yet to really solidify. The degree of mission criticality of your application should be inversely related to the novelty of the technology you choose to implement.

On the other hand, we feel reasonably confident betting on this horse. Hibernate is not currently a J2EE specification it is a third-party ORM. But as mentioned in the previous chapter, Sun merged the EJB3 and JDO2 specification teams and invited the lead architects of Hibernate to sit on the team as well. It's a safe bet that this future specification, whatever it ends up being called, will bear more than a passing resemblance to Hibernate.

The examples in this chapter were designed to show Hibernate in the best possible light. Keep in mind, though, that the object model is simple and maps neatly to a single database table. We purposely show you the JDBC solution alongside the Hibernate solution to illustrate the fact that they both are viable options.

Bottom line: use common sense. Start with simple objects and see how it goes. You might be able to use Hibernate throughout your application without a hitch. Or you might bump up against a limitation early in the process. Only you can decide how Hibernate fits best into your development strategy.

◀ PREV

NEXT ▶



 PREV

NEXT 

5.2. Hibernate Mapping Files

At the core of Hibernate is the HBM mapping file. This XML file maps your object members to fields in a database table. Some might argue that the clever use of reflection could eliminate the need for this file by simply automatically mapping table fieldnames to class fields. While this is appealing, you might not have complete editorial control over the tables or the classes. By using a file, you have the flexibility to map any table field to any class field, regardless of the name.

Recall that our `CarDTO` has four fields: `id`, `make`, `model`, and `modelYear`. See how the `car.hbm.xml` file maps these fields to the `Car` table in [Example 5-1](#).

Example 5-1. car.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class
    name="com.jbossatwork.dto.CarDTO"
    table="CAR"  >

    <id
      name="id"
      column="ID"
      type="int" >

      <generator class="native" />
    </id>

    <property
      name="make"
      type="java.lang.String"
      column="MAKE" />

    <property
      name="model"
      type="java.lang.String"
      column="MODEL" />

    <property
      name="modelYear"
      type="java.lang.String"
      column="MODEL_YEAR" />

  </class>
</hibernate-mapping>
```

- The `<class>` element matches POJO to table. It is possible to map a single POJO to multiple tables and vice versa, but we'll stick with the simple use case for this example.
- The `<id>` element identifies the Primary-Key/Unique Identifier field. The `<generator>` element tells Hibernate how the PK is created. "Native" tells hibernate to rely on the underlying database to generate the key. There are many different types of generators: "assigned" allows the program to specify the unique value. Use this when the PK has a specific meaning, such as a phone or social security number. Another common generator type is "increment," which lets Hibernate generate its own sequence number. (Recall that you set up an auto-incrementing Primary Key field in Hypersonic by using the "identity" keyword: "CREATE TABLE CAR (ID BIGINT identity, MAKE VARCHAR(50)...);")

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

5.3. Hibernate MBean Service Descriptor

Now that we have the HBM file in place, we must create an MBean service configuration file for Hibernate. Hibernate is a service, no different than Hypersonic or any of the others. Each MBean needs a service configuration file like [Example 5-4](#) so that JBoss will recognize it and run it on startup.

Example 5-4. hibernate-service.xml

```
<server>
  <mbean code="org.jboss.hibernate.jmx.Hibernate"
         name="jboss.har:service=Hibernate">
    <attribute name="DatasourceName">java:/JBossAtWorkDS</attribute>
    <attribute name="Dialect">
      org.hibernate.dialect.HSQLDialect</attribute>
    <attribute name="SessionFactoryName">
      java:/hibernate/SessionFactory</attribute>
    <attribute name="CacheProviderClass">
      org.hibernate.cache.HashtableCacheProvider
    </attribute>
  </mbean>
</server>
```

Let's step through it line by line.

- The <mbean> element names the service and specifies the implementing class.
- The <attribute name="DatasourceName"> element is a link to your Hypersonic datasource using the global JNDI name.
- The <attribute name="Dialect"> element tells Hibernate which type of database it talks to. As much as we'd like to believe the "s" in SQL stands for "standard," the acronym is short for "Structured Query Language." Each database vendor's implementation of SQL varies, and this setting allows Hibernate to generate well-formed SQL for the specific database in question. Other common dialects include org.hibernate.dialect.Oracle9Dialect and org.hibernate.dialect.MySQLDialect.
- The <attribute name="SessionFactoryName"> element is the global JNDI name for this service's SessionFactory. We'll use this name in *jboss-web.xml* (and map it to a local ENC-style name in *web.xml*).
- Finally, the <attribute name="CacheProviderClass"> element tells Hibernate what caching strategy to use. Rather than going round trip to the database for each request, Hibernate caches the results to improve performance. (See the Hibernate documentation for a more in-depth discussion of the different CacheProviders.)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

5.4. Creating a HAR

Now that we created the HBM files and the Hibernate MBean service deployment descriptor, we are ready to bundle things up and deploy it as a part of the EAR.

Hibernate applications are bundled up in a Hibernate Archive (HAR). We use the standard Ant `<jar>` task in [Example 5-5](#) to create the HAR.

Example 5-5. build.xml

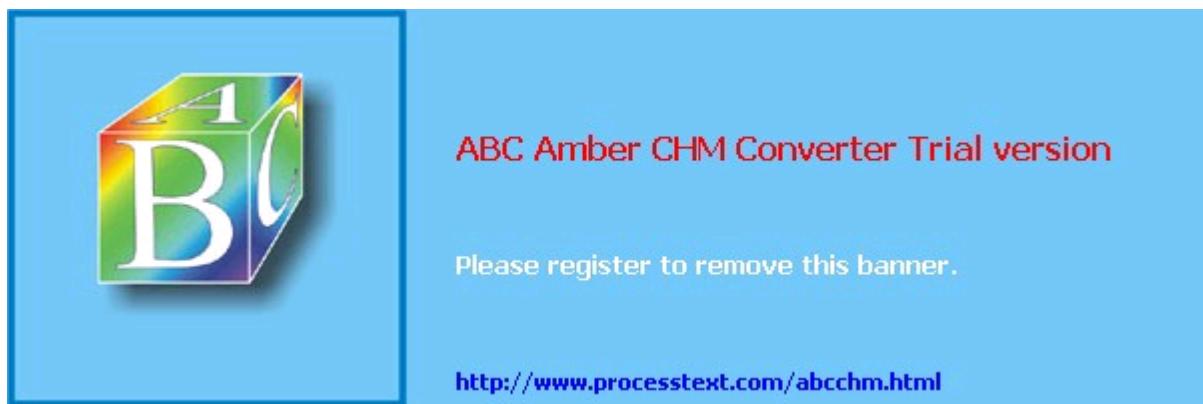
```
<!-- ===== -->
<target name="har" depends="generate-hbm"
       description="Builds the Hibernate HAR file">
    <mkdir dir="${distribution.dir}" />

    <jar destfile="${distribution.dir}/jaw.har">
        <!-- include the hbm.xml files -->
        <fileset dir="${gen.source.dir}">
            <include name="**/*.hbm.xml"/>
        </fileset>

        <!-- include hibernate-service.xml -->
        <metainf dir="${hibernate.dir}">
            <include name="hibernate-service.xml"/>
        </metainf>
    </jar>
</target>
```

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)[NEXT ▶](#)

5.5. Adding the HAR to the EAR

Moving up a level from the `common` subproject to the `master` project, we need to make sure that the EAR file includes our newly created HAR. The master build file already handles this step, since the `jaw.har` file is built in the same directory as `common.jar`. (Look in `common/build/distribution` to confirm this.)

But what about our `application.xml` file? This is traditionally where we identify the JARs that are included in the EAR. HARs are not a standard part of a J2EE EAR file, so JBoss looks for a `jboss-app.xml` file to handle container-specific exceptions to the standard. `src/META-INF/` stores this file, right next to your `application.xml` file. [Example 5-6](#) shows what it looks like.

Example 5-6. jboss-app.xml

```
<!DOCTYPE jboss-app PUBLIC "-//JBoss//DTD J2EE Application 1.4//EN"
          "http://www.jboss.org/j2ee/dtd/jboss-app_4_0.dtd">
<jboss-app>
  <module>
    <har>jaw.har</har>
  </module>
</jboss-app>
```

Our `<ear>` task now includes this file along with the traditional `application.xml` in [Example 5-7](#).

Example 5-7. Master build.xml

```
<!-- ===== -->
<target name="ear" depends="compile"
      description="Packages all files into an EAR file">
  <mkdir dir="${build.dir}" />
  <mkdir dir="${distribution.dir}" />

  <echo message="##### Building EAR #####" />
  <ear destFile="${distribution.dir}/${ear.name}"
       appxml="${meta-inf.dir}/application.xml" >
    <!-- files to be included in / -->
    <fileset dir="${webapp.war.dir}" />
    <fileset dir="${common.jar.dir}" />

    <!-- include jboss-app.xml -->
    <metainf dir="${meta-inf.dir}">
      <include name="jboss-app.xml"/>
    </metainf>

  </ear>
</target>
```

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

5.6. Creating a JNDI Lookup

Hibernate is now ready to use. Let's move to the `webapp` subproject to create the necessary JNDI lookups.

The first step toward using it is creating a JNDI reference to it in `jboss-web.xml`. [Example 5-8](#) shows what the file looks like now.

Example 5-8. jboss-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 2.3v2//EN"
 "http://www.jboss.org/j2ee/dtd/jboss-web_3_2.dtd">

<jboss-web>

<resource-ref>
    <res-ref-name>jdbc/JBossAtWorkDS</res-ref-name>
    <jndi-name>java:/JBossAtWorkDS</jndi-name>
</resource-ref>

<resource-ref>
    <res-ref-name>hibernate/SessionFactory</res-ref-name>
    <jndi-name>java:/hibernate/SessionFactory</jndi-name>
</resource-ref>

</jboss-web>
```

Remember that `<res-ref-name>` is the local ENC-style name. With the implied `java:comp/env/` prefix, the full ENC-style JNDI name for our Hibernate service is `java:comp/env/hibernate/SessionFactory`.

The global JNDI name (`<jndi-name>`) matches the setting in `hibernate-service.xml` `java:/hibernate/SessionFactory`.

[Example 5-9](#) shows what the `web.xml` file now looks like.

Example 5-9. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <servlet>
        <servlet-name>Controller</servlet-name>
        <servlet-class>com.jbossatwork.ControllerServlet</servlet-class>

    </servlet>

    <servlet-mapping>
        <servlet-name>Controller</servlet-name>
        <url-pattern>/controller/*</url-pattern>
    </servlet-mapping>

    <resource-ref >
        <res-ref-name>jdbc/JBossAtWorkDS</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)

5.7. Hibernate Checklist

Before we get to the actual `HibernateCarDAO`, let's recap what we've done up to this point. We:

- Added Hibernate tags to the CarDTO and created an Ant task to create the Mapping File (`cardto.hbm.xml`)
- Created a Hibernate MBean service descriptor (`hibernate-service.xml`)
- Bundled these two files up in a HAR (`common/build/distribution/jaw.har`)
- Created a `jboss-app.xml` file so JBoss would know how to deploy the HAR included in the EAR
- Created a global JNDI reference to the Session Factory in `jboss-web.xml` (`java:/hibernate/SessionFactory`)
- Created a local JNDI reference to the Session Factory in `web.xml` conforming to the J2EE ENC naming style (`java:comp/env/hibernate/SessionFactory`)
- Modified the ServiceLocator class that encapsulates all JNDI lookups to return a Hibernate Session

JBoss 4.0.2 and Hibernate 3.0.2 Issues

Bugs are facts of life. Even though JBoss and Hibernate are excellent products, JBoss 4.0.2 and Hibernate 3.0.2 straight out of the box have one significant problema core JAR file that inadvertently was left out of the distribution. The Apache Jakarta Commons Collections JAR needs to be downloaded separately and installed for Hibernate to work correctly. Visit the website (http://jakarta.apache.org/site/downloads/downloads_commons-collections.cgi) and download Version 2.1.1 of the JAR. Then copy it to one of the following directories:

- `$JBOSS_HOME/server/default/lib`
- `$JBOSS_HOME/server/default/deploy/jboss-hibernate.deployer`

This issue should be resolved in JBoss 4.0.3.

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

5.8. HibernateCarDAO

It's taken a while to get here, but now that the infrastructure is in place, we can get to the whole point of this chapter—seeing Hibernate in action. In [JDBCCarDAO](#), we performed the SQL query and manually marshaled the `ResultSet` rows into `CarDTO` objects.

[Example 5-13](#) shows what the JDBC code looks like.

Example 5-13. JDBCCarDAO.java

```
private static final String DATA_SOURCE="java:comp/env/jdbc/JBossAtWorkDS";

public List findAll( )
{
    List carList = new ArrayList( );
    DataSource dataSource = null;
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    try
    {
        dataSource = ServiceLocator.getDataSource(DATA_SOURCE);
        conn = dataSource.getConnection( );
        stmt = conn.createStatement( );
        rs = stmt.executeQuery("select * from CAR");

        while(rs.next( ))
        {
            CarDTO car = new CarDTO( );
            car.setId(rs.getInt("ID"));
            car.setMake(rs.getString("MAKE"));
            car.setModel(rs.getString("MODEL"));
            car.setModelYear(rs.getString("MODEL_YEAR"));
            carList.add(car);
        }

    }
    catch (Exception e)
    {
        System.out.println(e);
    }
    finally
    {
        try
        {
            if(rs != null){rs.close( );}
            if(stmt != null){stmt.close( );}
            if(conn != null){conn.close( );}
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

return carList;
}
```

It's nearly 50 lines of code, if you include all the exception handling. The point of showing this to you is to remind you that we do all the work by hand.

Now let's see the same method implemented with Hibernate in [Example 5-14](#).

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

5.9. Adding a Car

To add a new car, we'll create a new link on the `viewCarList` page, as in [Figure 5-2](#).

Figure 5-2. viewCarList with Add Car link

[Add Car]

Make	Model	Model Year
Toyota	Camry	2005
Toyota	Corolla	1999
Ford	Explorer	2005

This link will submit an `addCar` action request to our `ControllerServlet` (<http://localhost:8080/jaw/controller?action=addCar>).

The `ControllerServlet` in [Example 5-16](#) places an empty `CarDTO` in the `Request` scope and redirects to the `carForm.jsp` page.

Example 5-16. ControllerServlet.java

```
// perform action
    if(VIEW_CAR_LIST_ACTION.equals(actionName))
    {
        CarDAO carDAO = new HibernateCarDAO();
        request.setAttribute("carList", carDAO.findAll());
        destinationPage = "/carList.jsp";
    }
    else if(ADD_CAR_ACTION.equals(actionName))
    {
        request.setAttribute("car", new CarDTO());
        destinationPage = "/carForm.jsp";
    }
    else
    {
        String errorMessage = "[" + actionName + "] is not a valid
action.";
        request.setAttribute(ERROR_KEY, errorMessage);
    }
}
```

The `carForm.jsp` page in [Figure 5-3](#) allows the user to type in the details of a new car and save it.

Figure 5-3. carForm.jsp

[Return to List]

Make	<input type="text"/>
Model	<input type="text"/>
Model Year	<input type="text"/>
<input type="button" value="Save"/>	<input type="button" value="Reset"/>

[Example 5-17](#) shows the JSP code.

Example 5-17. carForm.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

```
<html>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

5.10. Editing a Car

To edit an existing car, we'll add another set of links on the `viewCarList` page in [Figure 5-4](#).

Figure 5-4. Editing cars

[Add Car]

Action	Make	Model	Model Year
Edit	Toyota	Camry	2005
Edit	Toyota	Corolla	1999
Edit	Ford	Explorer	2005
Edit	Honda	Accord	2006

The `Edit` links each contain the ID of the displayed car. They call the `ControllerServlet` using the `editCar` action. [Example 5-21](#) shows the JSP code.

Example 5-21. carList.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>

<head>
    <link rel="stylesheet" type="text/css" href="default.css">
</head>

<body>
    <p><a href="controller?action=addCar">[Add Car]</a></p>

    <table>
        <tr>
            <th>Action</th>
            <th>Make</th>
            <th>Model</th>
            <th class="model-year">Model Year</th>
        </tr>

        <c:forEach items='${carList}' var='car'>
        <tr>
            <td><a href="controller?action=editCar&id=${car.id}">Edit</a></td>
            <td>${car.make}</td>
            <td>${car.model}</td>
            <td class="model-year">${car.modelYear}</td>
        </tr>
        </c:forEach>

    </table>
</body>

</html>
```

The `ControllerServlet` in [Example 5-22](#) catches the request, queries the requested `CarDTO` out of the database using the ID, places it in the `Request` scope, and finally redirects to the `carForm.jsp`.

Example 5-22. ControllerServlet.java

```
// perform action
    if(VIEW_CAR_LIST_ACTION.equals(actionName))
    {
        CarDAO carDAO = new HibernateCarDAO( );
    }
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

5.11. Deleting a Car

Hopefully, at this point you are getting into a groove. As you incrementally add functionality to the website, you will tend to complete the same steps in order each time:

- Modify the View (JSP pages).
- Modify the Controller (`ControllerServlet`).
- Modify the Model (`CarDTO`, `CarDAO`).

Let's add the final bit of functionality for this chapter allowing the user to delete cars. First, we'll modify the View in [Figure 5-6](#).

Figure 5-6. carList.jsp with delete

[Add Car]

Delete	Action	Make	Model	Model Year
<input type="checkbox"/>	Edit	Toyota	Camry	2005
<input checked="" type="checkbox"/>	Edit	Toyota	Corolla	1999
<input type="checkbox"/>	Edit	Ford	Explorer	2005
<input checked="" type="checkbox"/>	Edit	Honda	Accord	2006
<input type="button" value="Delete Checked"/> <input type="button" value="Reset"/>				

The final `carList.jsp` in [Example 5-26](#) allows the user to check individual cars and delete them in bulk. Notice that clicking on the `Delete` column header checks all of the records. Clicking the "Reset" button unchecks all records.

Example 5-26. carList.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>

<head>
    <link rel="stylesheet" type="text/css" href="default.css">

    <script language="JavaScript">
        function checkAll(field)
        {
            for (i=0; i < field.length; i++)
            {
                field[i].checked = true;
            }
        }
    </script>
</head>

<body>
    <p><a href="controller?action=addCar">[Add Car]</a></p>

    <form name="deleteForm" method="post" action="controller">
        <input type="hidden" name="action" value="deleteCar" />
        <table>
            <tr>
                <th><a href="javascript:checkAll(document.deleteForm.id)">Delete</a></th>
                <th>Action</th>
                <th>Make</th>
                <th>Model</th>
                <th class="model-year">Model Year</th>
            </tr>

```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

5.12. Looking Ahead...

Hopefully you feel like we're picking up a head of steam here. You should feel comfortable with the Web tier, as well as the Persistence tier, at this point. The next several chapters focus on the middle tier (the Business tier), where you will get comfortable with Stateless Session Beans (SLSBs) and Message-Driven Beans (MDBs).

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

Chapter 6. Stateless Session Beans

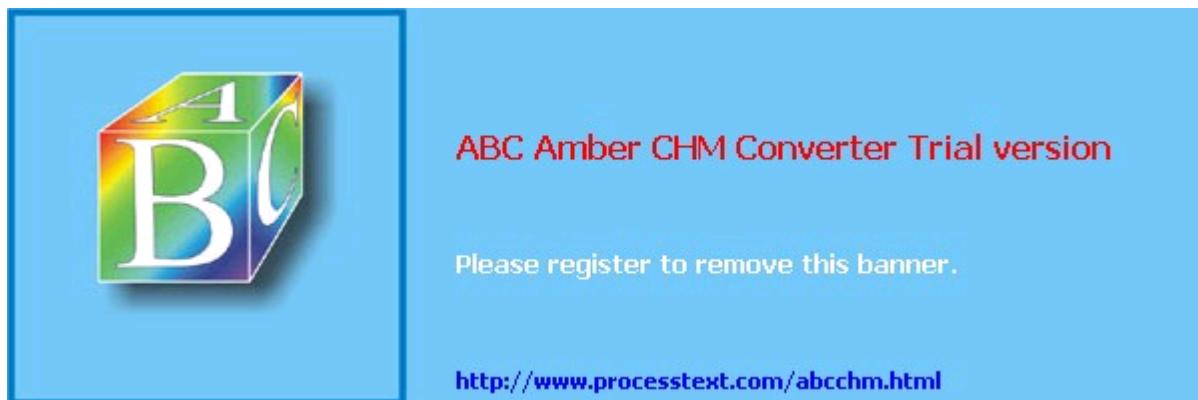
Everything we've done up to this point technically could have been done without using JBoss. Our web tier simply uses the embedded Tomcat Servlet container we could run our WAR unchanged in a standalone Tomcat instance. Our persistence tier used both JDBC and Hibernate. Again, both are available in non-JBoss installations. We simply took advantage of them because they come bundled with JBoss.

We now are firmly in the middle of the business tier. The next several chapters will focus on components that must be run in JBoss. More specifically, we'll look at EJB components that must run inside an EJB container.

Many people would argue that Enterprise JavaBeans are what put the "E" in the Java2 Enterprise Edition. But you also could argue that web-centric applications that use only JSPs and Servlets are still legitimate J2EE applications. These next sections will introduce you to technologies that allow you to build a large-scale, distributed, transaction-based Enterprise application.

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

6.1. Issues with EJBs

Our two biggest complaints about EJBs are that:

- They require a complex set of programming artifacts for deployment.
- Developers tend to overuse them.

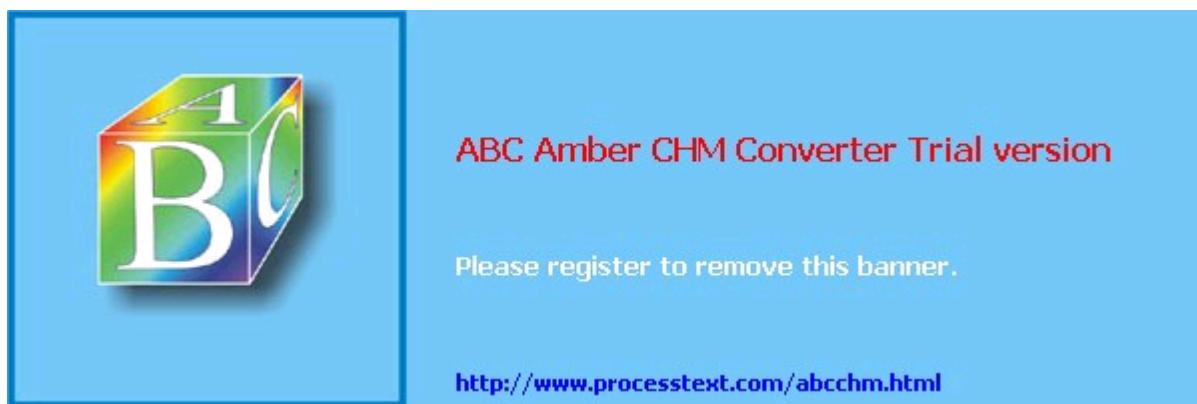
The complexity of the EJB programming model has hindered the adoption of J2EE. To deploy an EJB, you are dealing with as many as seven or more files (five Java files and two deployment descriptors). As you'll see later on, the main thrust of J2EE 1.5 (which includes EJB 3.0) is to simplify and lighten J2EE development by eliminating many of the programming artifacts required to deploy components. J2EE 1.5 will still offer core enterprise-class services such as transaction management and security, but it will be much easier to use. This book will not go into much detail about EJB 3.0 because the specification hasn't been finalized. But it will briefly discuss the improvements you'll see once EJB 3.0 becomes available. This book uses EJB 2.1, and it still has to deal with deployment descriptors and extra Java interfaces. It will show how XDoclet generates most of the code for us so that we can focus on business logic.

As software developers, most of us have the tendency to over-engineer a solution. You may also have heard the old adage, "If all you've got is a hammer, every problem tends to look like a nail." Bruce Tate, author of *Better, Faster, Lighter Java* (O'Reilly), calls the combination of these two predilections the "Golden Hammer" theory. Developers who know only EJBs tend to use them in wholly inappropriate situations.

Tate's book, as the title might imply, argues against the notion that every application you develop should be "Enterprise-grade," with all the associated complexity and overhead. He recommends the use of other third-party ORMs rather than CMP Entity Beans, and also suggests using a "better, faster, lighter" container to manage transactional applications called the Spring Framework. Like Hibernate, Spring is not J2EE-specification compliant and can run inside either a J2EE container or standalone. Both Hibernate and Spring have had a huge impact on the upcoming EJB 3.0 specification, and due to the influence of these external frameworks, EJB 3.0 will be lighter and simpler to use.

◀ PREV

NEXT ▶



 PREV

NEXT 

6.2. Should I Use EJB or Not?

From their inception, EJBs have been used for the right and wrong reasons. We've seen EJB-based projects that have failed miserably and others that were highly successful. So what made the difference? Here are a couple of key success factors with EJBs:

- Developer knowledge and maturity
- Decision-making

EJB development and deployment is complex with EJB 2.1 and earlier, so even if you have a good reason for using EJBs, your project still could fail if your development team hasn't worked with them before. If you're in this situation, take the same approach with EJBs as you would with any other technology that's new to you learn how to use it before you build your system. The best way to quickly get up to speed on EJB is to get some training and to augment your team with a few experienced developers who've used them to develop production-quality systems.

But taking classes, hiring experienced developers, and reading books aren't enough. You also should consider some guidelines to help you decide if EJBs are appropriate for your architecture. As we said earlier, there are both good and bad reasons for using EJBs (of all types).

Here are some reasons for not using EJBs:

Your application uses threads

The EJB specification forbids using threads because the threads your code creates are outside the container's control and could cause unexpected results.

Your application is read-only or read-mostly

If your application reads data from your database most or all of the time, then transactions aren't important to you. In this case, EJBs are overkill.

Here are some dubious or wrong reasons for developing with EJBs:

You want to use database connection pooling with JDBC DataSources

You don't need an EJB or an EJB container to use a [DataSource](#). You can configure a standalone Tomcat container with a database connection pool and use a [DataSource](#).

You want to use Entity Beans for persistence

Other third-party ORMs, such as Hibernate, provide more functionality and better performance.

You think you'll need them later, so you're adding them now

Don't over-architect. EJBs add complexity, so add them only when they help you solve a problem. Remember that you can always go back and refactor to an EJB solution when the need arises.

You want to group your related business logic together

You could group order-related functionality into an EJB, but you could also do this with a POJO. The need to group related logic together isn't a strong enough reason to use an EJB. But if your business processes require transactions or if you have external remote clients, you should consider using an EJB see the good reasons below.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

6.3. Business Tier

Let's review the three basic tiers of J2EE development. The Web tier handles the look and feel of the application. It is what the user interacts with. The Persistence tier handles long-term data storage. The Business tier handles coarse-grained business rules.

For example, consider the "Add Car" user story that we implemented in the last chapter. The user clearly has a way to enter new cars into the systemthe Web tier provides an HTML form. The new car has a way to be persisted for the long-termour DAOs in the Persistence tier. But what about the business rules for adding a new car to the JAW inventory?

When a dealer gets a new car on the lot, many actions need to happen. Someone needs to physically receive the car. Someone else needs to affix the dealer logo to the back of the car. Accounting needs to add it to the books as an asset. Marketing needs to add it to the "New Arrivals" listing in the newspaper ads.

While these are actions that would be done by humans in the real world, the same types of things usually need to happen in a J2EE application as well. In a J2EE application, we call a grouping of activities a Transaction. Transactions should be atomicif one of the steps fails, all grouped activities should roll back to their initial state.

The classic example is a bank transactionif you transfer money from your checking account to your savings account, the two distinct activities are treated as an atomic transaction. If your `SavingsAccountDAO.depositMoney()` method fails, you want your `CheckingAccountDAO.withdrawMoney()` method call to roll back as well. Otherwise, the money you took out will be lost.

Transactions are usually synchronous and represent a series of sequential steps to carry out a business process. Another type of activity occurs over a longer course of timean asynchronous activity. These types of activities tend to be a series of individual steps that are performed in a specific order, but don't necessarily cause the previous step to roll back. Think "work-flow" instead of "transaction." Think macro-view instead of micro-view. For example, if marketing doesn't get the new car in this week's newspaper ad, we don't need to roll back the dealer logo application and the accounting activities.

The EJB specification provides specific technologies that handle both synchronous and asynchronous activities.

◀ PREV

NEXT ▶



[◀ PREV](#)

[NEXT ▶](#)

6.4. Enterprise JavaBeans

There are three types of EJBs:

Session Beans

Session Beans allow the developer to group the steps of a business process into a single transaction. The activities contained in a Session Bean are synchronous.

Message-Driven Beans (MDBs)

MDBs are good for processing business logic asynchronously and/or executing long-running tasks in the background. They listen on Java Messaging Service (JMS) destinations (Queues and Topics) and process incoming messages as they arrive.

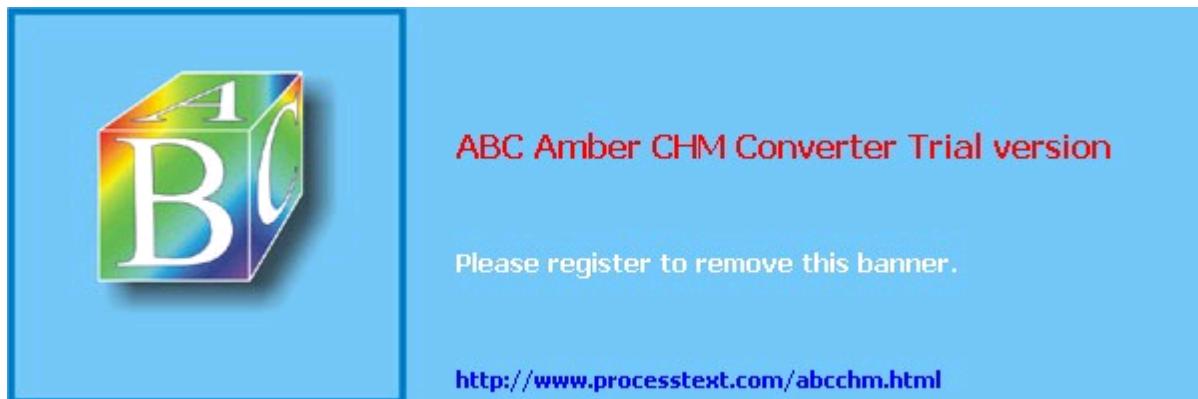
Entity Beans

Entity Beans are a persistence mechanism that encapsulates Create, Read, Update, and Delete (CRUD) operations on database tables.

Although some shops still use entity beans, most of the Java community has moved away from them in favor of other, more flexible third-party ORM solutions. However, the other two types of EJBs still are widely used. We'll talk about Session Beans in this chapter and MDBs in the next.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

6.5. Our Example

We are going to upgrade the JAW Motors application by adding a "Buy Car" user story that uses a transaction. In addition to marking a record into the `CAR` table as "`Sold`", we'll also insert a corresponding record into the `ACCOUNTING` table. A Session Bean is appropriate because buying a car is an atomic transaction. The update to the `CAR` table and the insert into the `ACCOUNTING` table must both succeed if one database operation fails, the other operation rolls back. We'll take three iterations to move from a web-only application to one that uses a Session Bean for its business logic:

Iteration 1

Introduce a Session Bean. The Controller Servlet uses the `InventoryFacadeBean` rather than the DAO to list the cars on the web page. All other actions in the Controller Servlet still use the DAO.

Iteration 2

We then move all business logic out of the Controller Servlet into the `InventoryFacadeBean` (which wraps the DAO).

Iteration 3

Upgrade the web pages, Controller Servlet, and `InventoryFacadeBean` to buy a car. We'll also create a new `AccountingHibernateDAO` and `AccountingDTO` for the `ACCOUNTING` table.

◀ PREV

NEXT ▶



 PREV

NEXT 

6.6. Iteration 1Introduce a Session Bean

In this Iteration, we'll take the following steps to introduce a Session Bean to the JAW Motors application and lay the groundwork for the rest of the chapter.

- Modify the Persistence Tier:
 - Add a `STATUS` column to the `CAR` table.
 - Make the `CarDTO` `Serializable`, and add a `status` field along with getter and setter methods.
 - Add a `filterByStatus()` method to the `HibernateCarDAO`.
- Upgrade the web site:
 - Refactor the Controller Servlet's `viewCarList` action to use the `InventoryFacadeBean` for finding all available (unsold) cars.
 - Add a `getEjbLocalHome()` method to the `ServiceLocator` to look up an EJB's Local Home Interface using JNDI.
 - Add EJB-based JNDI reference settings to the Web deployment descriptors (`web.xml` and `jboss-web.xml`).
 - Automate EJB-based JNDI reference settings in the Web deployment descriptors with XDoclet.
- Add a Session Bean:
 - Develop and deploy the `InventoryFacadeBean`.

6.6.1. Modifying the CAR Table

The `CAR` table's new `STATUS` column indicates whether a car is "Sold" or "Available." [Example 6-1](#) shows the new SQL in `ch06-a/sql/build.xml` used to create the table.

Example 6-1. sql/build.xml

```
CREATE TABLE CAR (
    ID BIGINT identity,
    MAKE VARCHAR(50),
    MODEL VARCHAR(50),
    MODEL_YEAR VARCHAR(50),
    STATUS VARCHAR(10)
);

INSERT INTO CAR (ID, MAKE, MODEL, MODEL_YEAR, STATUS)
VALUES (99, 'Toyota', 'Camry', '2005', 'Available');

INSERT INTO CAR (ID, MAKE, MODEL, MODEL_YEAR, STATUS)
VALUES (100, 'Toyota', 'Corolla', '1999', 'Available');

INSERT INTO CAR (ID, MAKE, MODEL, MODEL_YEAR, STATUS)
VALUES (101, 'Ford', 'Explorer', '2005', 'Available');
```

A car is considered "Available" when you create it. Now let's upgrade the `CarDTO`.

6.6.2. Upgrading the CarDTO with a Status Field

Here we'll make the `CarDTO` `Serializable` and add a new `status` data member along with corresponding setter and getter methods. [Example 6-2](#) shows the changes.

Example 6-2. CarDTO.java

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

6.7. Calling the Session Bean from the Controller Servlet

We're now going to introduce the `InventoryFacadeBean` to the JAW Motors application by calling it from the Controller Servlet. We'll modify the Controller Servlet so it uses the `InventoryFacadeBean` rather than the DAO to list the cars on the web page. For now, all other actions in the Controller Servlet still use the DAO. [Example 6-4](#) shows the changes.

Example 6-4. ControllerServlet.java

```
package com.jbossatwork;

...
import com.jbossatwork.ejb.*;
...
import javax.ejb.*;
...
public class ControllerServlet extends HttpServlet
{
    ...
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException
    {
        ...
        InventoryFacadeLocalHome inventoryHome;
        inventoryHome = (InventoryFacadeLocalHome)
            ServiceLocator.getEjbLocalHome(InventoryFacadeLocalHome.COMP_NAME);
        InventoryFacadeLocal inventory = null;
        try {
            inventory = inventoryHome.create();
        } catch (CreateException ce) {
            throw new RuntimeException(ce.getMessage());
        }
        // perform action
        if (VIEW_CAR_LIST_ACTION.equals(actionName))
        {
            request.setAttribute("carList", inventory.listAvailableCars());
            destinationPage = "/carList.jsp";
        }
        ...
    }
}
```

The `InventoryFacadeBean` is a JNDI-based resource, so we've encapsulated the JNDI lookup with the `ServiceLocator`. We'll show the look up code in detail in the next section. The `ServiceLocator.getEjbLocalHome()` call does a JNDI lookup on the EJB's Local Home, and creates a Local Home Interface. The `viewCarList` action now calls the `InventoryFacade`'s `listAllAvailableCars()` method to find all available (unsold) cars in the inventory.

Now that we've shown how to invoke the `InventoryFacadeBean`, let's take a closer look at the `ServiceLocator` that wraps the JNDI lookup.

6.7.1. Factoring Out the JNDI Calls

We've used the `ServiceLocator` throughout this book to wrap JNDI lookup calls. [Example 6-5](#) is the new EJB-related method.

Example 6-5. ServiceLocator.java

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

6.8. EJB-Based JNDI References in Web-Based Deployment Descriptors

In previous chapters we've used the `web.xml` file to describe and deploy Servlets and JNDI resources. [Example 6-6](#) shows the new EJB-based JNDI references in `web.xml` so we can use the `InventoryFacade` EJB from the web tier.

Example 6-6. `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  ...
  <ejb-local-ref>
    <ejb-ref-name>ejb/InventoryFacadeLocal</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>InventoryFacadeLocalHome</local-home>
    <local>InventoryFacadeLocal</local>
  </ejb-local-ref>
  ...
</web-app>
```

The `<ejb-local-ref>` element enables the web tier to access the `InventoryFacade` EJB through its Local Interface. The `<ejb-ref-name>` is the JNDI name for the EJB `java:comp/env/ejb/InventoryFacadeLocal`. Notice that you don't have to specify `java:comp/env/` because it is the assumed prefix. The `<ejb-type>` tells JBoss that this is a Session Bean. The `<local-home>` and `<local>` elements respectively specify the class name of the `InventoryFacade` EJB's Local Home and Local Component Interfaces.

A JNDI resource is linked into an application only if we ask for it. JBoss binds resources under its in-JVM context, `java:/`. The `jboss-web.xml` file provides a mapping between the J2EE-style ENC names and the local JBoss-specific JNDI names that JBoss uses to deploy JNDI-based resources. [Example 6-7](#) shows the EJB-related JNDI references in `jboss-web.xml`.

Example 6-7. `jboss-web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 2.4//EN"
 "http://www.jboss.org/j2ee/dtd/jboss-web_4_0.dtd">

<jboss-web>
  ...
  <ejb-local-ref>
    <ejb-ref-name>ejb/InventoryFacadeLocal</ejb-ref-name>
    <local-jndi-name>InventoryFacadeLocal</local-jndi-name>
  </ejb-local-ref>
  ...
</jboss-web>
```

The `jboss-web.xml` descriptor maps the J2EE-style JNDI names to JBoss-specific JNDI names. The `<ejb-local-ref>` element defines a local reference to the `InventoryFacade` Bean. The textual value of each `<ejb-ref-name>` element in `jboss-web.xml` MUST match the value of an `<ejb-ref-name>` in `web.xml`. The JNDI name in `<ejb-ref-name>` is relative to `java:comp/env`, so the full JNDI name is what we want: `java:comp/env/ejb/InventoryFacadeLocal`. The JNDI name in `<local-jndi-name>` is the name JBoss uses internally to reference the EJB for local access.

6.8.1. Automating EJB-Related JNDI Settings in Web-Based Deployment Descriptors

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

6.9. Session Bean Types

There are two types of Session Beans:

Stateful Session Beans

Hold on to conversational state and maintain a long-running dialog with a client. A Shopping Cart is an example.

Stateless Session Beans

Do NOT hold on to conversational state, so the client must pass all data needed by the Stateless Session Bean's business methods.

There is nothing in the example that requires us to maintain state between Session Bean calls. You'll probably find that most transactions will end up being Stateless. Here are some of the differences between Stateful and Stateless Session Beans :

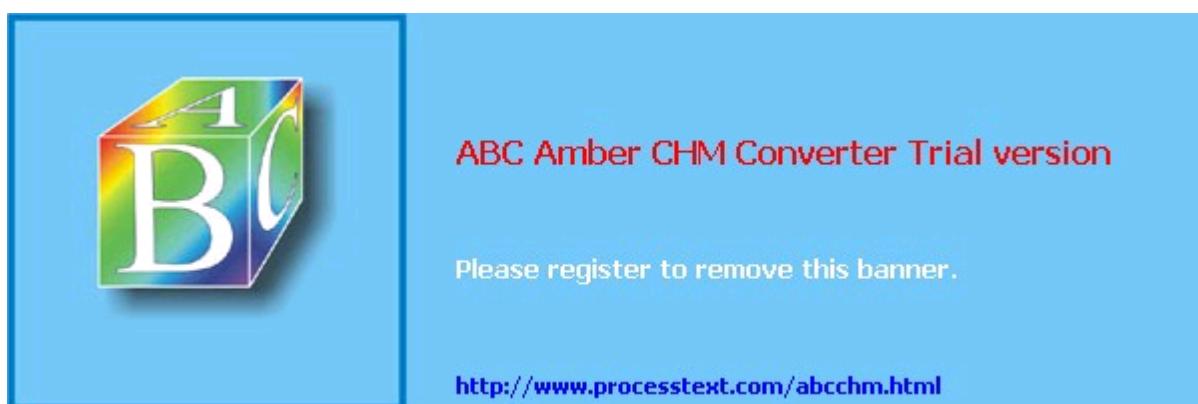
- Stateful Session Beans maintain internal state, which causes significant overhead. Stateless Session Beans are lightweight and do not hold onto application-specific data members.
- A Stateful Session Bean is tied to a single client, so the container creates a new instance for each client that invokes the Bean's `create()` method. A Stateless Session Bean instance is *not* tied to a client, so Stateless Session Beans are more scalable because they are reusableeach instance can service multiple clients concurrently.
- Stateless Session Beans are never passivated (swapped out of the container's memory and into secondary storage) because there is no need to restore internal state, but a container can passivate a Stateful Session Bean, incurring significant I/O overhead.

Due to performance reasons, Stateful Session Beans have fallen into disuse, and we recommend using Stateless Session Beans.

There is really no difference in deploying a Stateful or Stateless Session Beanyou specify the Session Bean type in the `ejb-jar.xml` deployment descriptor. But before concerning ourselves with deployment, let's look at how to implement a Session Bean.

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

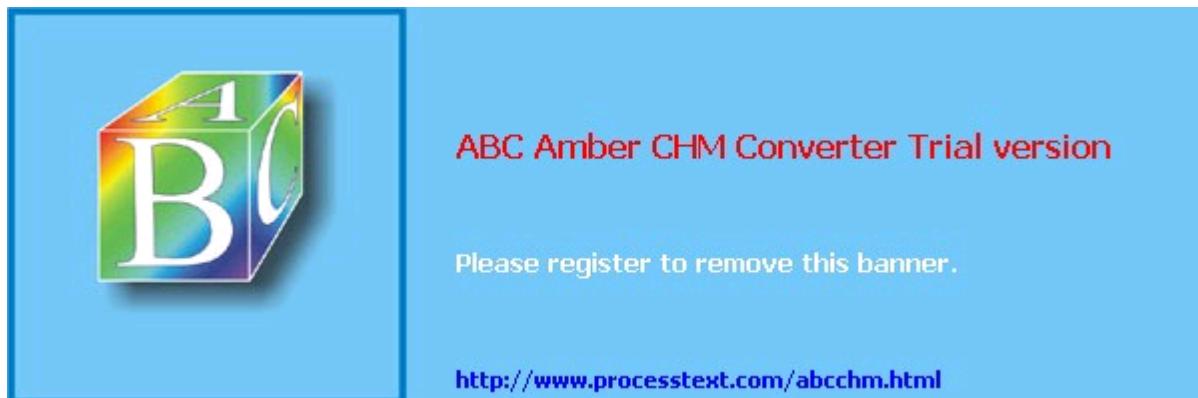
6.10. Session Beans

To add a Session Bean to the JAW Motors application, we need to do the following:

- Define the Local and Remote Interface(s).
- Create the Home Interface(s).
- Develop the Bean Class Code.
- Deploy the Bean with EJB deployment descriptors.
- Automate the Bean deployment with Ant and XDoclet.
- Create the EJB JAR file with Ant.
- Add the EJB JAR file to the EAR.
- Register the EJB JAR file in `application.xml`.
- Copy the EJB JAR file into the EAR.

◀ PREV

NEXT ▶



 PREV

NEXT 

6.11. Remote Versus Local EJB Calls

Since we're creating a `buyCar()` method, it could be called from any number of clients. We happen to be writing a web application, but the same transaction could conceivably be called from a Swing Application or even a web service.

This method is highly cohesiveit does only one thingbuy a car from our inventory. It is also loosely coupled to the other tiersthose there is nothing that returns HTML that would tie it to the web tier, and since it calls a DAO, nothing ties it to a specific database or persistence strategy.

We now need to decide how the other tiers are going to make this `buyCar()` method call. The Servlet container is co-located on the same server in the same JVM as the EJB container, so we can use the Session Bean's Local interfaces. All objects are in the same memory space, so under the covers, JBoss will pass things around by reference (technically, the memory location of the objects).

As our application grows, we might decide to split out this functionality to physically separate servers. We might move our web tier outside the firewall, move the Persistence tier (or at least the database server) to a box of its own, and leave the EJBs on a third box in the middle. Once we've made this step, we can grow each tier to a cluster of serversa cluster of Tomcat servers, DB servers, and JBoss servers.

Once we move to separate boxes, we need to start making Remote calls to our EJB instead of local calls. Since our Servlets and EJBs are no longer in the same JVM and memory space, the container can no longer make calls by reference. It now has to make calls by valuethis is done by serializing the object, streaming it over the network from one box to the other and reconstituting it on the other side.

Another reason why you might create remote interfaces is if you are writing a Swing client. Since your application is truly distributed, your Swing client will need a way to make method calls on the remote server.

We'll create both local and remote interfaces so you can see what they look like. But remember YAGNIif you don't have any immediate plans to run on separate boxes, don't do it. We'll just use local calls in our Servlets for this example.

Before we develop a Stateless Session Bean, let's explore the new directory structure we'll use for our EJB development environment

6.11.1. Exploring the New Directory Structure

In previous chapters, we had the following sub-directories:

common

Contains code and a `build.xml` file for building the Common JAR.

sql

Contains a `build.xml` file for creating the database.

webapp

Contains code and a `build.xml` file for building the WAR.

If you change to the `ch06/ch06-a` directory, you'll see that we've added an `ejb` sub-directorythis is our EJB development environment. The goal is to keep each portion of the application as autonomous as possible. Granted, most of the application will have dependencies on the `common` sub-project, but by providing individual Ant scripts we have the opportunity to build each portion of the project separately.

6.11.1.1. The ejb sub-project

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

6.12. Local and Remote Interfaces

The Local and Remote Interfaces define the business methods that an EJB exposes to its clients. These interfaces are sometimes called an EJB's Component Interfaces in EJB books and other related literature. An EJB's Local and Remote Interface methods serve the same purpose as public methods for a POJO.

6.12.1. Local Interface

The Local Interface defines an EJB's business methods accessed by local, or co-located, clients (Servlets, POJOs, or EJB) that run inside the container. When accessing an EJB through its local interface, there is no overhead for a network call and object serialization because everything runs in the same JVM. A client running outside the container cannot access a Session Bean using local interfaces. All methods in the Local Component Interface throw an `EJBException`. Our Local Interface, `InventoryFacadeLocal`, looks like [Example 6-9](#).

Example 6-9. InventoryFacadeLocal.java

```
package com.jbossatwork.ejb;

public interface InventoryFacadeLocal extends javax.ejb.EJBLocalObject
{
    public java.util.List listAvailableCars( ) throws javax.ejb.EJBException;
    public CarDTO findCar(int id) throws javax.ejb.EJBException;
    public void deleteCars(String[ ] ids) throws javax.ejb.EJBException;
    public void saveCar(CarDTO car) throws javax.ejb.EJBException;
    public void buyCar(int carId, double price) throws javax.ejb.EJBException;
}
```

6.12.2. Remote Interface

The Remote Interface defines an EJB's business methods accessed by remote clients applications that run outside the EJB container. All methods in the Remote Component Interface throw a `RemoteException`. Invoking an EJB through its remote interface incurs overhead for a network call and object serialization because the client and the EJB run in separate JVMs. [Example 6-10](#) is our Remote Interface, `InventoryFacadeRemote`.

Example 6-10. InventoryFacadeRemote.java

```
package com.jbossatwork.ejb;

public interface InventoryFacadeRemote extends javax.ejb.EJBObject
{
    public java.util.List listAvailableCars( ) throws
javax.ejb.RemoteException;
    public CarDTO findCar(int id) throws javax.ejb.RemoteException;
    public void deleteCars(String[ ] ids) throws javax.ejb.RemoteException;
    public void saveCar(CarDTO car) throws javax.ejb.RemoteException;
    public void buyCar(int carId, double price) throws
javax.ejb.RemoteException;
}
```

We've shown the Local and Remote Interfaces that define the EJB's business methods, but now we need to talk about the Home Interfaces.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

6.13. Home Interfaces

An EJB Home Interface is a factory that creates and removes EJB objects in an EJB container. A Session Bean's `create()` method initializes a new Session Bean and returns a proxy object so the client can start using the EJB. An EJB Home is analogous to a POJO's constructor, except that the EJB Home's `create` method gives you a client-side proxy rather than a concrete object. If you use a Local EJB Home, then you're using a Local Interface to call its business methods. If you look up a Remote EJB Home, then you're working with a Remote Interface to access a Bean's business methods.

6.13.1. Local Home Interface

The Local Home Interface defines an EJB's lifecycle methods that create and remove bean instances used by local, or co-located clients (Servlets, POJOs, or EJBs) that run inside the container. Our Local Home Interface, `InventoryFacadeLocalHome`, looks like [Example 6-11](#).

Example 6-11. InventoryFacadeLocalHome.java

```
package com.jbossatwork.ejb;

public interface InventoryFacadeLocalHome extends javax.ejb.EJBLocalHome
{
    public static final String
COMP_NAME="java:comp/env/ejb/InventoryFacadeLocal";
    public static final String JNDI_NAME="InventoryFacadeLocal";

    public com.jbossatwork.ejb.InventoryFacadeLocal create()
        throws javax.ejb.CreateException;
}
```

This is a Session Bean, so you only need a simple `create()` method with an empty argument list.

6.13.2. Remote Home Interface

The Remote Home Interface defines an EJB's lifecycle methods that create and remove bean instances used by applications outside the container. [Example 6-12](#) is our Remote Home Interface, `InventoryFacadeRemoteHome`.

Example 6-12. InventoryFacadeRemoteHome.java

```
package com.jbossatwork.ejb;

public interface InventoryFacadeRemoteHome extends javax.ejb.EJBHome
{
    public static final String COMP_NAME="java:comp/env/ejb/InventoryFacade";
    public static final String JNDI_NAME="InventoryFacadeRemote";

    public com.jbossatwork.ejb.InventoryFacadeRemote create()
        throws javax.ejb.CreateException, java.rmi.RemoteException;
}
```

6.13.3. The Bean Class

The Bean Class provides the implementation for a Session Bean's business methods. Our Bean Class, `InventoryFacadeBean`, looks like [Example 6-13](#).

Example 6-13. InventoryFacadeBean.java

```
package com.jbossatwork.ejb;
```

```
import java.util.*;
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

6.14. Reviewing Iteration 1

It's taken a while to get here, but we now have the core infrastructure in place to use EJBs. The main purpose of Iteration 1 was to introduce an EJB to the JAW Motors application. So, to review :

- We modified the Persistence Tier so that we can tell if a car is available or sold by:
 - Adding a `STATUS` column to the `CAR` table.
 - Making the `CarDTO` `Serializable`, and adding a `status` field along with getter and setter methods.
 - Adding a `filterByStatus()` method to the `HibernateCarDAO`.
- We upgraded the web site by:
 - Refactoring the Controller Servlet's `viewCarList` action to use the `InventoryFacadeBean` for finding all available (unsold) cars.
 - Adding a `getEjbLocalHome()` method to the `ServiceLocator` to look up an EJB's Local Home Interface using JNDI.
 - Adding EJB-based JNDI reference settings to the Web deployment descriptors (`web.xml` and `jboss-web.xml`).
 - Automating EJB-based JNDI reference settings in the Web deployment descriptors with XDoclet.
- We added `InventoryFacade` Session Bean by:
 - Defining the Local and Remote Interface(s).
 - Creating the Home Interface(s).
 - Developing the Bean Class Code.
 - Deploying the Bean with EJB deployment descriptors.
 - Automating the Bean deployment with Ant and XDoclet so that XDoclet created the Remote, Remote Home, Local, and Local Home interfaces for us. We also used XDoclet to create the `ejb-jar.xml` and `jboss.xml` deployment descriptors.
- We added EJBs to our EAR file by:
 - Creating the EJB JAR file with Ant.
 - Registering the EJB JAR file in `application.xml`.
 - Copying the EJB JAR file into the EAR.

As you've just seen, adding EJBs to a J2EE application is non-trivial. But you can mitigate this overhead with Ant and XDoclet.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

6.15. Testing Iteration 1

Now that we've written and deployed an EJB and called it from the Controller Servlet, let's test the application to ensure that everything still works properly. Here are the steps to build and deploy the application:

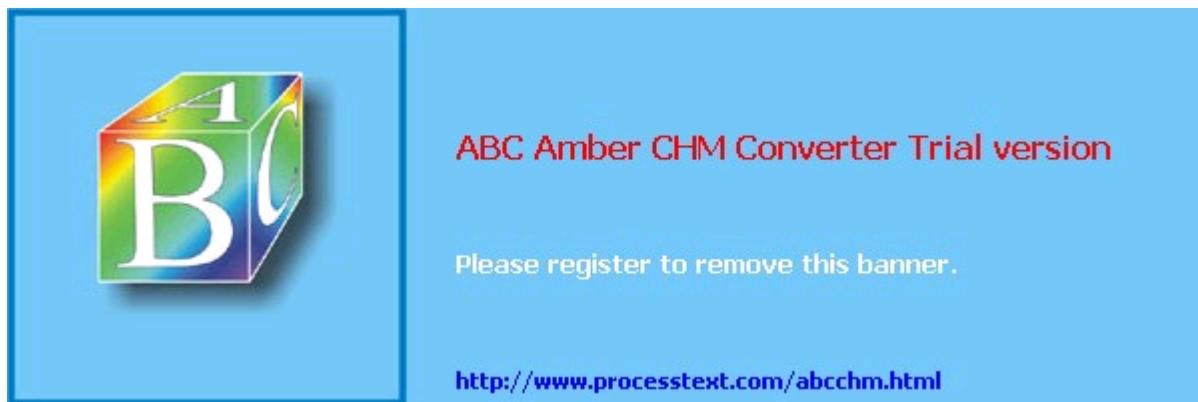
- Type `ant` in the root directory of `ch06-a` to build the project.
- Shutdown JBoss so that the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBOSS_HOME/server/default/deploy` directory.
- Start JBoss back up.
- Go to the `ch06-a/sql` sub-directory and type `ant` to modify the database.
- Visit <http://localhost:8080/jaw> in a web browser.

When you click on the "View Inventory" link on the JAW Motors home page, the Controller Servlet takes you to the Inventory page where you can view, add, edit, or delete cars in JAW Motors' Inventory.

We've spent a lot of time in Iteration 1 establishing the core infrastructure for using EJBs in the application, and now we'll move on to Iterations 2 and 3, where we'll move the business logic from the Web Tier to the EJB, and then finally add the ability to buy a car. Since we have all the plumbing in place, these next two Iterations will be much shorter than the first.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

6.16. Iteration 2Move Business Logic Out of the Controller

In this Iteration, we're going to move all business logic out of the Controller Servlet into the `InventoryFacadeBean` (which groups our synchronous activities together and wraps the DAO). We'll take the following steps:

- Move the code from the Controller Servlet's actions into the `InventoryFacadeBean`.
- Modify the Controller Servlet to call the new methods in the `InventoryFacadeBean`.
- Modify the `HibernateCarDAO` to work with CMT.
- Change the `ServiceLocator`'s `getHibernateSession()` method to work with CMT.

6.16.1. Refactoring the Business Logic

If you'll recall from earlier in this chapter, we had modified the Controller Servlet so its `viewCarList` action used the `InventoryFacadeBean` to find all available cars. We will now refactor the Controller Servlet by moving all the business logic from its actions into the `InventoryFacadeBean`. [Example 6-21](#) shows the modifications to the Controller Servlet.

Example 6-21. ControllerServlet.java

```
package com.jbossatwork;

...
import com.jbossatwork.ejb.*;
...
import javax.ejb.*;
...
public class ControllerServlet extends HttpServlet
{
    ...
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException
    {
        ...
        InventoryFacadeLocalHome inventoryHome;
        inventoryHome = (InventoryFacadeLocalHome)
            ServiceLocator.getEjbLocalHome(InventoryFacadeLocalHome.COMP_NAME);

        InventoryFacadeLocal inventory = null;

        try {
            inventory = inventoryHome.create();
        } catch (CreateException ce) {
            throw new RuntimeException(ce.getMessage());
        }

        // perform action
        if (VIEW_CAR_LIST_ACTION.equals(actionName))
        {
            request.setAttribute("carList", inventory.listAvailableCars());
            destinationPage = "/carList.jsp";
        }
        else if (ADD_CAR_ACTION.equals(actionName))
        {
            request.setAttribute("car", new CarDTO());
            destinationPage = "/carForm.jsp";
        }
        else if (EDIT_CAR_ACTION.equals(actionName))
        {
            int id = Integer.parseInt(request.getParameter("id"));
            ...
        }
    }
}
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

6.17. Reviewing Iteration 2

In Iteration 2, we moved all business logic out of the Controller Servlet into the `InventoryFacadeBean` (which groups our synchronous activities and wraps the DAO). We took the following steps:

- Moved the business logic from the Controller Servlet's actions into the `InventoryFacadeBean`.
- Modified the Controller Servlet to call the new methods in the `InventoryFacadeBean`.
- Modified the `HibernateCarDAO` to work with CMT by removing the transaction setup and tear down code. We also removed the code that closes Hibernate Sessions.
- Changed the `ServiceLocator`'s `getHibernateSession()` method to work with CMT by calling the `SessionFactory`'s `getCurrentSession()` method.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

6.18. Testing Iteration 2

Now that we've moved all business logic from Controller Servlet to the `InventoryFacadeBean`, let's test our application to ensure that everything still works properly. Here are the steps to build and deploy the application:

- Type `ant` in the root directory of `ch06-b` to build the project.
- Shut down JBoss so the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBOSS_HOME/server/default/deploy` directory.
- Start JBoss back up.
- Visit <http://localhost:8080/jaw> in a web browser.

When you click on the "View Inventory" on the JAW Motors home page, the Controller Servlet takes you to the Inventory page where you can view, add, edit, or delete cars in JAW Motors' Inventory.

Now that we have all the infrastructure in place, let's move on to Iteration 3 where we finally add the ability to buy a car.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

6.19. Iteration 3Buy a Car

In this Iteration, we're going to enable a user to buy a car from the JAW Motors web site. We'll add a new page to enable the user to buy a car. When the user presses the "Submit" button on the new "Buy Car" page, the business logic changes the car's status to "Sold" and inserts a new row into the ACCOUNTING table.

We'll take the following steps:

- Upgrade the web site:
 - Add a "Buy Car" link to the Car Inventory page (`carList.jsp`).
 - Add a "Buy Car" action to the Controller Servlet.
- Modify the Persistence Tier:
 - Create an ACCOUNTING table.
 - Write a `HibernateAccountingDTO`.
 - Develop a `HibernateAccountingDAO`.
- Change the Session Bean:
 - Add a new `buyCar()` method to the `InventoryFacadeBean` that encapsulates a Container-Managed Transaction that involves both the CAR and ACCOUNTING tables.

6.19.1. Upgrade the Web Site: Adding a "Buy Car" Link

We've added a "Buy Car" link to the JAW Motors Car Inventory page (`carList.jsp`) as shown in [Figure 6-2](#).

Figure 6-2. JAW Motors Inventory Page
[Add Car]

Delete	Action	Make	Model	Model Year	Buy Car
<input type="checkbox"/>	Edit	Toyota	Camry	2005	Buy
<input type="checkbox"/>	Edit	Toyota	Corolla	1999	Buy
<input type="checkbox"/>	Edit	Ford	Explorer	2005	Buy

[Delete Checked](#) [Reset](#)

When the user clicks on the "Buy" link, the Controller routes them to the Buy Car page as depicted in [Figure 6-3](#) so they can buy the car.

The user enters her price in the form and presses the "Save" button. The Controller Servlet then takes the user back to the Car Inventory page. The purchased car is no longer available, so it won't show up on the Car Inventory page.

Figure 6-3. JAW Motors Buy Car Page

Car	2005 Toyota Camry
Price	<input type="text"/>
Save	Reset

Now that the web pages are done, we have to add actions to the Controller Servlet for buying a car. [Example 6-27](#) shows the changes.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

6.20. The AccountingDTO

The `AccountingDTO` is an object that Hibernate persists to the `ACCOUNTING` table. The `AccountingDTO` has four fields: `id`, `carId`, `price`, and a `saleDate`. The `AccountingDTO` has getter and setter methods for each data member, along with XDoclet tags telling the `<hibernate>` XDoclet task how to generate an HBM mapping file that maps between the `AccountingDTO` object and the `ACCOUNTING` table. The `AccountingDTO` looks very similar to the `CarDTO` from previous chapters, so we're not showing the code here. If you want to see the `AccountingDTO` code, you can find it in the `com.jbossatwork.dto` package under the `common` sub-directory.

We've created the `AccountingDTO`, so now we need to develop the `HibernateAccountingDAO` to persist the `AccountingDTO` to the `ACCOUNTING` table.

◀ PREV

NEXT ▶



6.21. Developing the HibernateAccountingDAO

The `HibernateAccountingDAO` looks similar to the `HibernateCarDAO`, but instead of writing all the CRUD, we only need to insert a row into the `ACCOUNTING` table. Here, we have only a single `create()` method. [Example 6-29](#) shows the code for the `HibernateAccountingDAO`.

Example 6-29. `HibernateAccountingDAO.java`

```
package com.jbossatwork.dao;

import java.util.*;
import org.hibernate.*;
import org.hibernate.criterion.*;
import com.jbossatwork.dto.AccountingDTO;
import com.jbossatwork.util.*;

public class HibernateAccountingDAO implements AccountingDAO
{
    private static final String HIBERNATE_SESSION_FACTORY =
        "java:comp/env/hibernate/SessionFactory";

    public HibernateAccountingDAO( ) { }

    public void create(AccountingDTO accountingData)
    {
        Session session = null;

        try
        {
            session = ServiceLocator.getHibernateSession(
                HIBERNATE_SESSION_FACTORY);

            session.save(accountingData);
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

The `create()` method calls the `ServiceLocator` to get the Hibernate Session, and inserts a new row into the `ACCOUNTING` table by calling the Session's `save()` method for the `AccountingDTO`. Just like we did before, we let the container manage the transaction, and we're not closing the Session.

We've added all the infrastructure to the web site and the Persistence Tier so we can buy a car. We now wrap up by adding a `buyCar()` method to the `InventoryFacadeBean`.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)

6.22. Adding buyCar() to the InventoryFacadeBean

Example 6-30 is the `InventoryFacadeBean`'s `buyCar()` method.

Example 6-30. `InventoryFacadeBean.java`

```
...
public class InventoryFacadeBean implements SessionBean {
    ...
    /**
     * @ejb.interface-method
     * @ejb.transaction
     * type="Required"
     *
     */
    public void buyCar(int carId, double price) throws EJBException {
        CarDAO carDAO = new HibernateCarDAO();
        CarDTO car;
        AccountingDAO accountingDAO = new HibernateAccountingDAO();
        AccountingDTO accountingData;

        car = carDAO.findById(carId);
        car.setStatus(CarDTO.STATUS_SOLD);
        carDAO.update(car);

        accountingData = new AccountingDTO(carId, price);
        accountingDAO.create(accountingData);
    }
    ...
}
```

The `buyCar()` method encapsulates a transaction for buying a car. This method takes the `carId` and `price` supplied by the caller to mark a car as "Sold" in the `CAR` table and record the sale in the `ACCOUNTING` table. If either the update to the `CAR` table or the `ACCOUNTING` table fails, the container rolls everything back. If all activities complete successfully, then all changes are committed to the database. After instantiating the DAOs, we use the `CarDAO` to find the car using the `carId`. To mark the car as "sold", we set the `CarDTO`'s status to "Sold" and call the `HibernateCarDAO`'s `update()` method to update the car's status in the `CAR` table. To record the sale, we instantiate a new `AccountingDTO` with the `carId` and `price`, and then call the `HibernateAccountingDAO`'s `create()` method to insert a new row in the `ACCOUNTING` table.

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

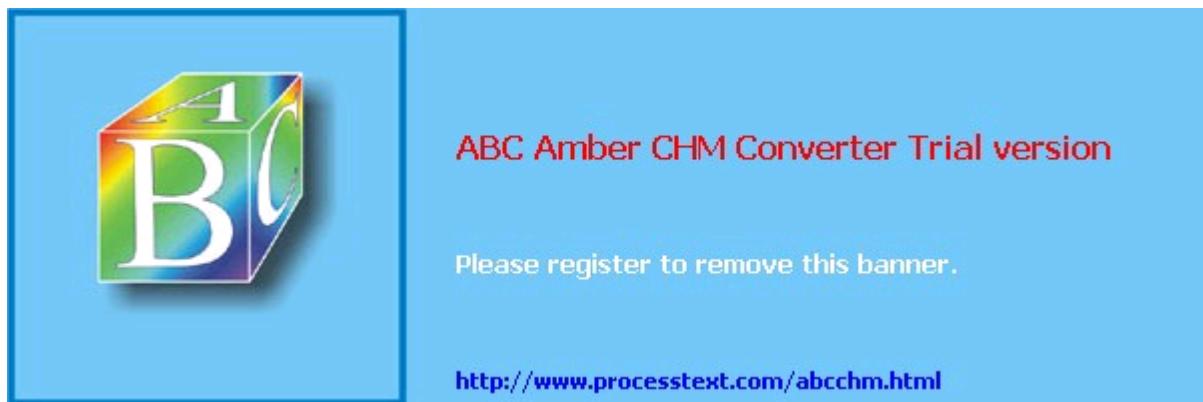
6.23. Reviewing Iteration 3

Before moving on to test buying a car, let's recap what we did in Iteration 3:

- Upgraded the web site:
 - Added a "Buy Car" link to the Car Inventory page (`carList.jsp`).
 - Added a "Buy Car" action to the Controller Servlet.
- Modified the Persistence Tier:
 - Created an `ACCOUNTING` table.
 - Wrote a `HibernateAccountingDTO`.
 - Developed a `HibernateAccountingDAO`.
- Changed the Session Bean:
 - Added a new `buyCar()` method to the `InventoryFacadeBean` that encapsulated a Container-Managed Transaction that involved the `CAR` and `ACCOUNTING` tables.

[◀ PREV](#)

[NEXT ▶](#)



6.24. Testing Iteration 3

Now that we've developed all the code and infrastructure for buying a car, let's test the application to ensure that everything works properly. Here are the steps to build and deploy the application:

- Type `ant` in the root directory of `ch06-c` to build the project.
- Shut down JBoss so the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBoss_HOME/server/default/deploy` directory.
- Start JBoss back up.
- Go to the `ch06-c/sql` sub-directory and type `ant` to modify the database.
- Visit <http://localhost:8080/jaw> in a web browser.

When you click on the "View Inventory" on the JAW Motors home page, the Controller Servlet takes you to the Inventory page where you can view, add, edit, delete, or buy cars in JAW Motors' Inventory. Click on one of the "Buy" links and the Controller will route you to the Buy Car page. Enter a price in the form and press the "Save" button. The Controller Servlet then takes you back to the Car Inventory page. The car that was just purchased is no longer available, so it won't show up on the Car Inventory page.

As a final test, we need to ensure that the transaction was recorded properly in the database. Go to the `ch06-c/sql` sub-directory and type: "`ant query`". The `query` target queries the `CAR` and `ACCOUNTING` tables. Depending on which car you bought, you should see something like this on the command console:

```
query:
[echo] Checking the CAR and ACCOUNTING tables ...
[sql] Executing commands
[sql] ID,MAKE,MODEL,MODEL_YEAR,STATUS
[sql] 99,Toyota,Camry,2005,Sold
[sql] 100,Toyota,Corolla,1999,Available
[sql] 101,Ford,Explorer,2005,Available

[sql] 0 rows affected
[sql] ID,CAR_ID,PRICE,SALE_DATE
[sql] 0,99,12000.0,2005-06-01

[sql] 0 rows affected
[sql] 2 of 2 SQL statements executed successfully
```



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

6.25. Final Thoughts on Session Beans

EJBs are the foundation for the rest of this book, so we took a lot of time to explain when and when not to use them, and how to deploy them. Here are the key take-away points on EJBs.

- Use EJBs when you really need them.
- Know why you're using EJBs.
- We use Stateless Session Beans for managing transactions and Message-Driven Beans for asynchronous processing.
- EJBs require a lot of extra work, but you only have to do the setup once:
 - Adding a new EJB Ant build script that uses XDoclet and creates the EJB JAR file.
 - Referencing the EJB JAR in `application.xml`.
 - Adding the EJB JAR file to the EAR.
- After the initial setup, things get better. The extra programming artifacts required by EJB 2.1 (Home and Component Interfaces and the deployment descriptors) are tedious and error-prone, but you can generate everything with XDoclet.
- Under EJB 2.1, you still have to implement the callback methods by hand.
- With EJB 3.0, the programming artifacts and callback methods go away. So, you're left with a business interface and a simple POJO that contains only the business methods that you care about. EJB 3.0 will streamline EJB development and deployment. We hope that a simpler EJB spec will encourage developers and architects to consider EJBs for their future needs when it makes sense to use them.
- We moved our business logic from the Controller Servlet into the `InventoryFacade` Stateless Bean for the following reasons:
 - Running our code from within an EJB enables us to use Container-Managed Transactions (CMT), which greatly reduces the amount of code you have to write.
 - Since we may not always have a web client, we want other types of clients to use our encapsulated business logic as services. We can expose some of the `InventoryFacadeBean`'s methods as Web Serviceswe'll show you how to do this in the Web Services chapter.
- When using Hibernate 3 from within an EJB that uses CMT, remember to do the following:
 - Make sure that your EJB method runs within the scope of a transaction.
 - Get a Hibernate Session by using the Session Factory's `getCurrentSession()` method.
 - The Hibernate transaction API calls are no longer needed because the container manages the transactions.
 - Never close your Hibernate Session because doing this loses your changeslet the container do it for you.
- The `InventoryFacadeBean`'s `buyCar()` method encapsulated a Container-Managed Transaction that involved both the `CAR` and `ACCOUNTING` tables.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

6.26. Looking Ahead ...

This chapter discussed reasons for and against using EJBs. We added the ability to buy a car on the JAW Motors web site by introducing an `InventoryFacade` Stateless Session Bean that encapsulates business logic and uses CMT to manage transactions. Along the way, we introduced expected improvements with EJB 3.0, showed the relationship between Hibernate 3 and CMT, and deployed our new Session Bean on JBoss.

In the next chapter, we'll add an MDB to the JAW Motors Application.

◀ PREV

NEXT ▶



Chapter 7. Java Message Service (JMS) and Message-Driven Beans

If you've worked through all the previous chapters, you have a fully functional vertical slice of the JAW Motors application that allows you to view, add, edit (update), delete, and buy cars. JAW Motors now wants to run a credit check on potential customers to pre-qualify them for car loans as part of the auto buying process. Like many other businesses, JAW Motors doesn't do its own credit verification and uses an external online service to run a credit check. Once we add a form to the JAW Motors web site to verify someone's credit, what happens when they enter their credit information and press the submit button? Should we make them wait until the external service finishes? This process could take several minutes, hours, or even days. Making the user wait around for the final result is unacceptable. After pressing the submit button, the user should be free to go back to what they were doingviewing, adding, editing, and deleting cars. We need a way to defer the credit verification process to the background so the user can continue using the web site.

J2EE provides the following mechanisms for deferring work to the background:

- JMS (Java Message Service)
- Message-Driven Beans (MDBs)

The JMS API provides a vendor-neutral standard programming interface for creating, sending, and receiving messages. JMS enables Java applications to asynchronously exchange messages containing data and events. JMS 1.1 is part of the J2EE 1.4 standard.

Message-Driven Beans (MDBs) are EJBs that receive and process JMS messages. MDBs don't maintain state between invocations, so they resemble Stateless Session Beans. But MDBs differ from Stateless Session Beans in that:

- A client is completely decoupled from an MDBthe client sends a JMS message and the MDB picks up the message.
- After sending a JMS asynchronously, the client doesn't wait for the MDB to receive the message. The client continues executing other business logic.

Enterprise-class applications usually defer processing slow, expensive operations to run in the background so clients don't have to wait for completion. A UI sends an asynchronous JMS message, and an MDB receives the message and process it while the client can do other things in the foreground, such as interact with your application's web pages. On completion, the MDB could notify the client with a confirmation email, send another message, or update the database.

This chapter covers JMS and Message-Driven Beans. We'll upgrade the JAW Motors application by adding the ability to run a credit check on a customer by using JMS messaging and an MDB. To set realistic expectations, we won't invoke a real credit verification service, but instead will emulate the credit check as a long-running process. Along the way, we'll show how to deploy these technologies on JBoss.

Let's start by working down through the architecture.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

7.1. Sending Messages with JMS

To send JMS messages from the JAW Motors application, do the following:

- Upgrade the web site:
 - Add a "Run Credit Check" link and form.
 - Add a "Run Credit Check" action to the Controller Servlet.
- Add JMS:
 - Create an object that holds the user's credit information to send as a JMS message.
 - Write a utility class to encapsulate sending a JMS message.
 - Add JMS-based JNDI reference settings to the Web-based deployment descriptors ([web.xml](#) and [jboss-web.xml](#)).
 - Automate JMS-based JNDI reference settings with XDoclet.
 - Deploy a JMS Queue on JBoss with an MBean.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

7.2. Upgrade the Site: Running a Credit Check

We've added a "Run Credit Check" link to the JAW Motors homepage, as shown in [Figure 7-1](#).

Figure 7-1. JAW Motors homepage

JAW Motors

[View Inventory](#)
[Run Credit Check](#)

When the user clicks on the "Run Credit Check" link, the Controller routes them to the Run Credit Check page as depicted in [Figure 7-2](#).

Figure 7-2. JAW Motors Run Credit Check page

[\[Return to Main Page\]](#)

Name	<input type="text"/>
SSN	<input type="text"/>
Email	<input type="text"/>
<input type="button" value="Submit Credit Info"/> <input type="button" value="Reset"/>	

The user enters his name, Social Security Number (SSN), and email address in the form and presses the "Submit Credit Info" button. When a customer requests a credit check, he won't have to wait for the external credit verification process to complete. The Controller Servlet sends a JMS message asynchronously to back end components that process business logic for the request. The customer gets routed back to the JAW Motors home page and is then free to continue using the JAW Motors web site while the credit check completes in the background.

Now that the web pages are done, we have to add actions to the Controller Servlet for running the credit check. [Example 7-1](#) shows the changes.

Example 7-1. ControllerServlet.java

```
public class ControllerServlet extends HttpServlet
{
    ...
    private static final String VIEW_CREDIT_CHECK_FORM_ACTION =
        "viewCreditCheckForm";

    private static final String RUN_CREDIT_CHECK_ACTION = "runCreditCheck";
    ...

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        ...
        else if(VIEW_CREDIT_CHECK_FORM_ACTION.equals(actionName))
        {
            destinationPage = "/creditCheckForm.jsp";
        }
        else if(RUN_CREDIT_CHECK_ACTION.equals(actionName))
        {
            System.out.println("Credit Check:\nName = [" +
                request.getParameter("name") + "]");
            System.out.println("SSN = [" + request.getParameter("ssn") + "]");
            System.out.println("Email = [" + request.getParameter("email") +
                "]");
        }
    }
}
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

7.3. JMS Architecture Overview

We need to understand some of the basics of JMS architecture before using JMS in our application. Here are the players:

JMS Provider

A messaging system that implements the JMS specification. A JMS Provider, also known as a JMS Server, routes messages between clients.

Clients

Java applications that send or receive JMS messages. A message sender is called the Producer, and the recipient is called a Consumer.

Messages

Messages contain data or events exchanged between Producers and Consumers.

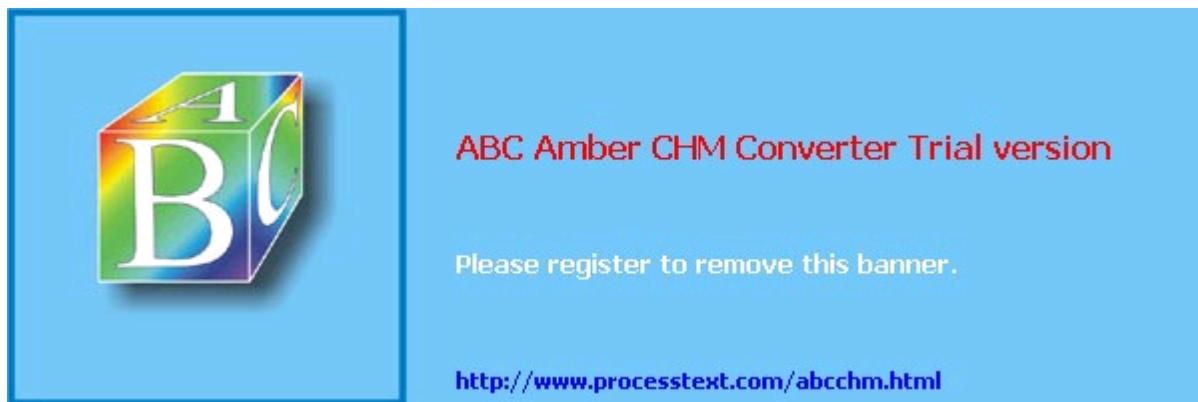
Destinations

A Producer sends a message to a JMS Destination (either a **Queue** or a **Topic**), and the Consumer(s) listening on the JMS Destination receives the message.

We now will cover messaging models to show how JMS routes messages between Producers and Consumers.

◀ PREV

NEXT ▶



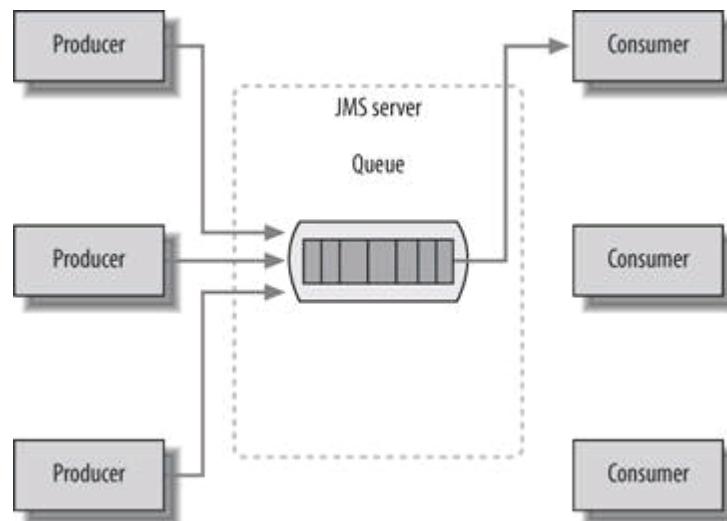
 PREV

NEXT 

7.4. JMS Messaging Models

JMS has two messaging models *Point-to-Point (P2P)* and *Publish-Subscribe (Pub-Sub)*. P2P is a traditional one-to-one queueing mechanism although multiple Consumers can listen on a queue, only one Consumer receives a particular message. Producers send messages to a queue, and the JMS Server delivers each message sequentially to a Consumer listening on the queue. [Figure 7-3](#) shows the relationships between Point-to-Point Producers and Consumers.

Figure 7-3. JMS Point-to-Point (P2P) Messaging model

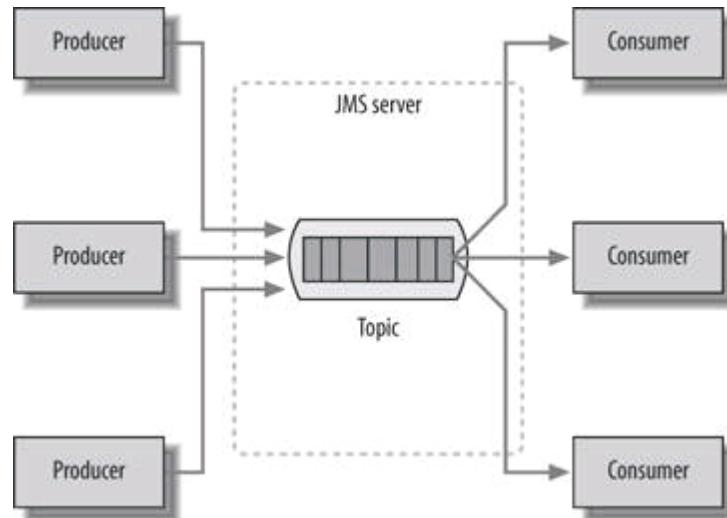


Publish-Subscribe is a one-to-many broadcast model, similar to a newsgroup or an RSS newsfeed. Producers publish messages to a topic, and the JMS Server delivers messages sequentially to those Consumers subscribed to that topic. [Figure 7-4](#) shows the relationships between Pub-Sub Producers and Consumers.

7.4.1. Choosing a Messaging Model

Most applications that use the Publish-Subscribe model could use Point-to-Point, and vice versa. How do you choose a messaging model? We'll give the standard consultant's answer "It depends." Just like other technologies, your application requirements determine how to use JMS messaging. According to Richard Monson-Haefel (author of the best-selling O'Reilly EJB book and co-author of O'Reilly's JMS book), Point-to-Point JMS messaging is like sending an email message to a single recipient. Point-to-Point is a way to distribute workload because only one consumer receives a particular message. Publish-Subscribe JMS messaging is like sending/broadcasting an email message to a recipient list many consumers receive the same message.

Figure 7-4. JMS Publish-Subscribe (Pub-Sub) Messaging model



◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

7.5. Creating a Message

Now we'll wrap the user's credit data in a `CreditCheckRequestDTO` object so we can send it as a JMS Message. [Example 7-2](#) shows the code.

Example 7-2. CreditCheckRequestDTO.java

```
package com.jbossatwork.dto;

import java.io.*;

public class CreditCheckRequestDTO implements Serializable
{
    private String name;
    private String ssn;
    private String email;

    public CreditCheckRequestDTO( ) { }

    public CreditCheckRequestDTO(String name, String ssn, String email)
    {
        this.name = name;
        this.ssn = ssn;
        this.email = email;
    }

    public String getName( )
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getSSN( )
    {
        return ssn;
    }

    public void setSSN(String ssn)
    {
        this.ssn = ssn;
    }

    public String getEmail( )
    {
        return email;
    }

    public void setEmail(String email)
    {
        this.email = email;
    }
}
```

The `CreditCheckRequestDTO` is similar to the `CarDTO` that you saw in previous chaptersit has setters and getters for each data member. To send an object as a JMS Message, it must obey the following rules:

- An object must implement `java.io.Serializable`.
- Each data member must be serializable. By default, `String`, the Java primitives (`int`,

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

7.6. Sending the Message

[Example 7-3](#) shows how to send a JMS Message from the Controller Servlet.

Example 7-3. ControllerServlet.java

```
public class ControllerServlet extends HttpServlet
{
    ...
    private static final String XA_QUEUE_CONNECTION_FACTORY =
        "java:comp/env/jms/MyXAQueueConnectionFactory";

    private static final String CREDIT_CHECK_QUEUE =
        "java:comp/env/jms/CreditCheckQueue";
    ...

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        ...
        else if(RUN_CREDIT_CHECK_ACTION.equals(actionName))
        {
            CreditCheckRequestDTO creditCheckReq = null;

            System.out.println("Credit Check:\nName = [" +
                request.getParameter("name") + "]");

            System.out.println("SSN = [" + request.getParameter("ssn") + "]");
            System.out.println("Email = [" + request.getParameter("email") +
                "]");

            creditCheckReq = new CreditCheckRequestDTO(
                request.getParameter("name"),
                request.getParameter("ssn"),
                request.getParameter("email"));

            JmsProducer.sendMessage(creditCheckReq,
XA_QUEUE_CONNECTION_FACTORY,
                CREDIT_CHECK_QUEUE);

            destinationPage = "/index.jsp";
        }
        ...
    }
    ...
}
```

If you'll recall from the "Running a Credit Check" section, pressing the "Submit Credit Info" from the "Run Credit Check" form invokes the Controller Servlet's `runCreditCheck` action. The Controller Servlet stores the user's credit information in the `CreditCheckReqDTO` and calls `JmsProducer.sendMessage()` to send the credit data as an asynchronous JMS message. After sending the message, the Controller Servlet returns the user to the main page. The `JmsProducer` is a utility class that hides the details of sending a JMS message—the next few sections cover the JMS API and the `JmsProducer` in greater detail.

We've encapsulated the user's credit information in a `CreditCheckRequestDTO` and shown how to send it as a JMS Message with the `JmsProducer`. After quickly reviewing the core JMS API, we'll see how the `JmsProducer` sends a JMS Message.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

7.7. Core JMS API

The JMS API resides in the `javax.jms` package. These are the most important classes and interfaces for our purposes:

Message

Holds business data and routing information. Although you'll find several types of `Messages`, most of the time you'll use either a `TextMessage` (that contains textual data in its body) or an `ObjectMessage` (that holds serializable objects in its body).

Destination

Holds messages sent by the Producer to be received by the Consumer(s). A Destination is either a `Queue` or a `Topic` that the JMS Server manages on behalf of its clients.

Connection

Enables a JMS client to send or receive `Messages`. Use a `Connection` to create one or more `Sessions`.

ConnectionFactory

A `ConnectionFactory` is either a `QueueConnectionFactory` or it is a `TopicConnectionFactory`, depending on the messaging model, and it exists to create `Connections`.

Session

A `Session` creates Producers, Consumers, and `Message`s. A Publish-Subscribe application uses a `TopicSession`, and a Point-to-Point application uses a `QueueSession`. `Sessions` are single-threaded.

◀ PREV

NEXT ▶



 PREV

NEXT 

7.8. Sending a JMS Message

Now that we know the basics of JMS, we'll take the following steps to send a message:

- Look up a `ConnectionFactory` using JNDI.
- Get a Connection from the `ConnectionFactory`.
- Create a Session associated with the Connection.
- Look up a Destination using JNDI.
- Create a Message Producer tied to the Destination.
- Create a Message.
- Send the Message.
- Tear down the Message Producer, Session, and Connection.

JMS requires a lot of low-level tedious API calls to send a message, so [Example 7-4](#) encapsulates everything into a `JmsProducer` utility object.

Example 7-4. JmsProducer.java

```
package com.jbossatwork.util;

import java.io.*;
import javax.jms.*;

/**
 * <code>JmsProducer</code> encapsulates sending a JMS Message.
 *
 */
public class JmsProducer {

    /**
     * Making the default (no arg) constructor private
     * ensures that this class cannot be instantiated.
     */
    private JmsProducer( ) { }

    public static void sendMessage(Serializable payload,
        String connectionFactoryJndiName, String destinationJndiName)
        throws JmsProducerException {

        try {
            ConnectionFactory connectionFactory = null;
            Connection connection = null;
            Session session = null;
            Destination destination = null;
            MessageProducer messageProducer = null;
            ObjectMessage message = null;

            connectionFactory = ServiceLocator.getJmsConnectionFactory(
                connectionFactoryJndiName);

            connection = connectionFactory.createConnection( );
            session = connection.createSession(false,
                Session.AUTO_ACKNOWLEDGE);
            destination =
                ServiceLocator.getJmsDestination(destinationJndiName);
            messageProducer = session.createProducer(destination);
        }
    }
}
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

7.9. JMS-Based JNDI References in Web-Based Deployment Descriptors

In previous chapters, we've used the `web.xml` file to describe and deploy Servlets and JNDI resources. [Example 7-6](#) shows the new JMS-based JNDI references in `web.xml`, so that we can use the JMS `ConnectionFactory` and the `CreditCheckQueue`.

Example 7-6. `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  ...
  <resource-ref>
    <res-ref-name>jms/CreditCheckQueue</res-ref-name>
    <res-type>javax.jms.Queue</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>

  <resource-ref>
    <res-ref-name>jms/MyXAQueueConnectionFactory</res-ref-name>
    <res-type>javax.jms.QueueConnectionFactory</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
  ...
</web-app>
```

`<res-ref-name>` is the JNDI name for each resource `java:comp/env/jms/CreditCheckQueue` and `java:comp/env/jms/MyXAQueueConnectionFactory`. Notice that you don't have to specify `java:comp/env/` because it is the assumed prefix. The `<res-type>` for the `CreditCheckQueue` is a JMS Queue, and `javax.jms.Queue` is its fully qualified class name. The `<res-type>` for the Connection Factory is a JMS Queue Connection Factory `javax.jms.QueueConnectionFactory`. We want JBoss to manage our JMS resources, so you set `<res-auth>` to `Container`.

A JNDI resource links into an application only if we ask for it. JBoss binds resources under its in-JVM context, `java:/`. The `jboss-web.xml` file provides a mapping between the J2EE-style ENC names and the local JBoss-specific JNDI names that JBoss uses to deploy JNDI-based resources. [Example 7-7](#) shows the JMS-related JNDI references in `jboss-web.xml`.

Example 7-7. `jboss-web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 2.4//EN"
"http://www.jboss.org/j2ee/dtd/jboss-web_4_0.dtd">

<jboss-web>
  ...
  <resource-ref>
    <res-ref-name>jms/CreditCheckQueue</res-ref-name>
    <jndi-name>queue/CreditCheckQueue</jndi-name>
  </resource-ref>

  <resource-ref>
    <res-ref-name>jms/MyXAQueueConnectionFactory</res-ref-name>
    <jndi-name>java:/JmsXA</jndi-name>
  </resource-ref>
  ...
</jboss-web>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

7.10. Deploying JMS Destinations on JBoss

JBoss implements and manages JMS Queues and Topics as JMX MBeans that create the Destination and register its JNDI name. Here are a few options for creating an MBean for our Queue:

The JMX console

Go to the JMX Console by visiting the following URL <http://localhost:8080/jmx-console>. Click on the `service=DestinationManager` link (under the `jboss.mq` heading) to see the JBossMQ `DestinationManager` MBean page. Look for the `createQueue()` operation and enter `jms/CreditCheckQueue` for the first parameter (the J2EE JNDI name would then be `java:comp/env/jms/CreditCheckQueue`), and `queue/CreditCheckQueue` for the second parameter (the JBoss-specific JNDI name). Using the JMX console is easy and provides a dynamic way to specify JMS resources, but your settings will be lost once you shut down JBoss.

Add an `<mbean>` element for the queue to

`$JBOSS_HOME/server/default/deploy/jms/jbossmq-destinations-service.xml` file.

This file is part of the core JBoss deployment and contains JBoss-specific default test Queues and Topics.

Create your own service descriptor file (post-fixed with `-service.xml`) that resides in `$JBOSS_HOME/server/default/deploy` and add an `<mbean>` element for the queue.

This is a custom file that you create either by hand or with an automated tool as part of your build and deployment process.

We created our own service file because we wanted your JMS destination settings to survive JBoss startup/shutdown, and we didn't want to co-mingle your application-specific JMS Destinations with JBoss-internal Destinations. Co-mingling destinations is bad because each time you upgrade to a new version of JBoss, you have to re-add the `<mbean>` elements to your service descriptor to make things work again. [Example 7-9](#) shows the `jaw-jms-destinations.xml` file that creates an MBean for the `CreditCheckQueue`.

Example 7-9. `jaw-jms-destinations.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<server>
    <mbean code="org.jboss.mq.server.jmx.Queue"
           name="jboss.mq.destination:service=Queue,name=CreditCheckQueue">

        <depends optional-attribute-name="DestinationManager">
            jboss.mq:service=DestinationManager
        </depends>
    </mbean>
</server>
```

To deploy our Queue to JBoss, the Ant build script copies the `jaw-jms-destinations-service.xml` file from the `ch07/src/META-INF` directory to the JBoss deployment directory `$JBOSS_HOME/server/default/deploy`.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

7.11. JMS Checklist

Before moving on to Message-Driven Beans, let's recap what we've done so far:

- Upgraded the web site:
 - Added a "Run Credit Check" link and form.
 - Added a "Run Credit Check" action to the Controller Servlet.
- Added JMS:
 - Created a `CreditCheckReqDTO` object that holds the user's credit information to send as a JMS message.
 - Wrote the `JmsProducer` utility class to encapsulate sending a JMS message.
 - Added JMS-based JNDI reference settings to the Web-based deployment descriptors (`web.xml` and `jboss-web.xml`).
 - Automated JMS-based JNDI reference settings with XDoclet.
 - Deployed the JMS `CreditCheckQueue` on JBoss with an MBean.

We developed the code to send a JMS message, and completely deployed your JMS Queue on JBoss. We now need to add a Message-Driven Bean that consumes and processes the credit check JMS message.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

7.12. Message-Driven Beans (MDBs)

It's taken a while to get here, but we now have the core infrastructure in place to send a JMS message. To consume JMS messages with an MDB, we'll do the following:

- Write the MDB.
- Deploy the MDB.
- Automate MDB deployment with XDoclet.

A Message-Driven Bean (MDB) is an EJB whose sole purpose is to consume JMS messages. When a JMS producer sends a message to a JMS destination, the MDB listening on that destination receives the message and processes it. MDBs are pooled, stateless, and do not have a Home or Component interface. An MDB implements the JMS `MessageListener` interface; its `onMessage()` method processes the message received from the JMS destination and implements business logic. Pooling enables concurrent behavior, so several instances of an MDB would run in parallel if multiple messages are in the queue or topic.

7.12.1. Writing an MDB

In the JAW Motors application, the `CreditCheckProcessor` MDB:

- Consumes a JMS `ObjectMessage` that contains a `CreditCheckRequestDTO`
- Invokes an emulated external credit verification service

[Example 7-10](#) is the code for the MDB.

Example 7-10. CreditCheckProcessorBean.java

```
package com.jbossatwork.ejb;

import javax.ejb.*;
import javax.jms.*;

import com.jbossatwork.dto.*;
import com.jbossatwork.util.*;

public class CreditCheckProcessorBean implements MessageDrivenBean,
MessageListener
{

    private MessageDrivenContext ctx = null;

    public CreditCheckProcessorBean( ) { }

    public void setMessageDrivenContext(MessageDrivenContext ctx)
throws EJBException {
        this.ctx = ctx;
    }

    /**
     * Required creation method for message-driven beans.
     */
    public void ejbCreate( ) {
        // no specific action required for message-driven beans
    }

    /** Required removal method for message-driven beans. */
    public void ejbRemove( ) {
        ctx = null;
    }
}
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

7.13. MDB Checklist

We took the following steps to add an MDB to the JAW Motors Application:

- Wrote the `CreditCheckProcessor` MDB.
- Deployed the `CreditCheckProcessor` MDB in `ejb-jar.xml` and `jboss.xml`.
- Automated deployment of the `CreditCheckProcessor` MDB with XDoclet.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)[NEXT ▶](#)

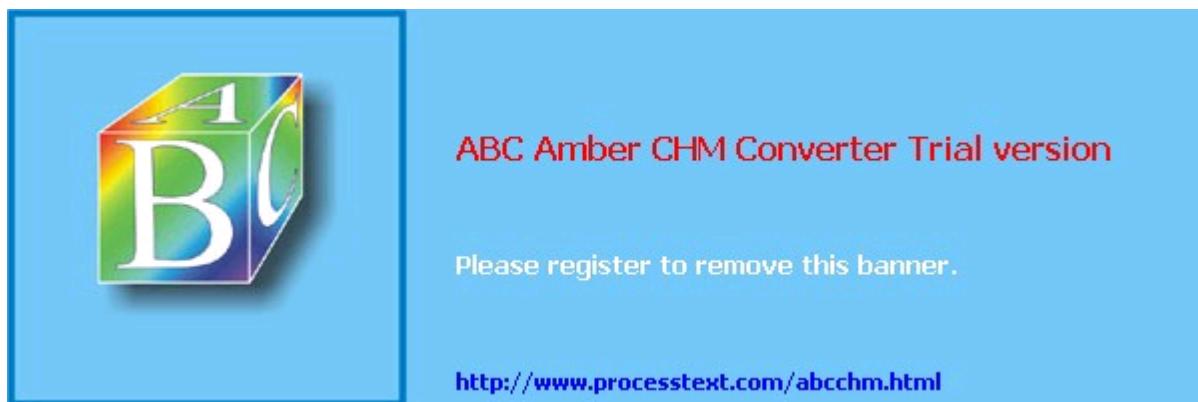
7.14. Testing the Credit Check

Now that we've deployed the JMS Queue and an MDB to run the credit check, let's test our application to ensure that everything still works properly. Here are the steps for building and deploying the application:

- Type `ant` in the root directory of `ch07` to build the project.
- Shut down JBoss so the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBoss_HOME/server/default/deploy` directory. Notice that the Ant build script also deploys the `jaw-jms-destinations.xml` JMS Destination service file to JBoss.
- Start JBoss back up.
- Visit <http://localhost:8080/jaw> in a web browser.

Click on the "Run Credit Check" link on the JAW Motors home page, enter data on the "Run Credit Check" form, and press the "Submit Credit Info" button. You should be routed back to the main page and see the following output on your JBoss console:

```
...
17:05:55,707 INFO  [STDOUT] CreditCheckProcessorBean.onMessage( ) : Received
message.
17:05:55,707 INFO  [STDOUT] Credit Check:
17:05:55,707 INFO  [STDOUT] Name = [Fred]
17:05:55,707 INFO  [STDOUT] SSN = [9999999999999999]
17:05:55,707 INFO  [STDOUT] Email = [fred@acme.org]
17:05:55,707 INFO  [STDOUT] Verifying Credit ...
17:05:59,703 INFO  [STDOUT] Credit Check Result = [Fail Credit Check]
```

[◀ PREV](#)[NEXT ▶](#)

◀ PREV

NEXT ▶

7.15. Looking Ahead ...

This chapter covered JMS and Message-Driven Beans. We upgraded the JAW Motors application by adding the ability to run a credit check on a customer by using JMS messaging and an MDB. Along the way, we showed how to deploy these technologies on JBoss.

The JBoss console output proves that the Controller Servlet sent the credit check JMS message and that the `CreditCheckProcessor` MDB consumed and processed the message. However, it isn't completely satisfying because the web site user doesn't know the final result of the credit verification process. In the next chapter, we'll upgrade the MDB to use the JavaMail API when sending the user an email notification message.

◀ PREV

NEXT ▶



[◀ PREV](#)

[NEXT ▶](#)

Chapter 8. JavaMail

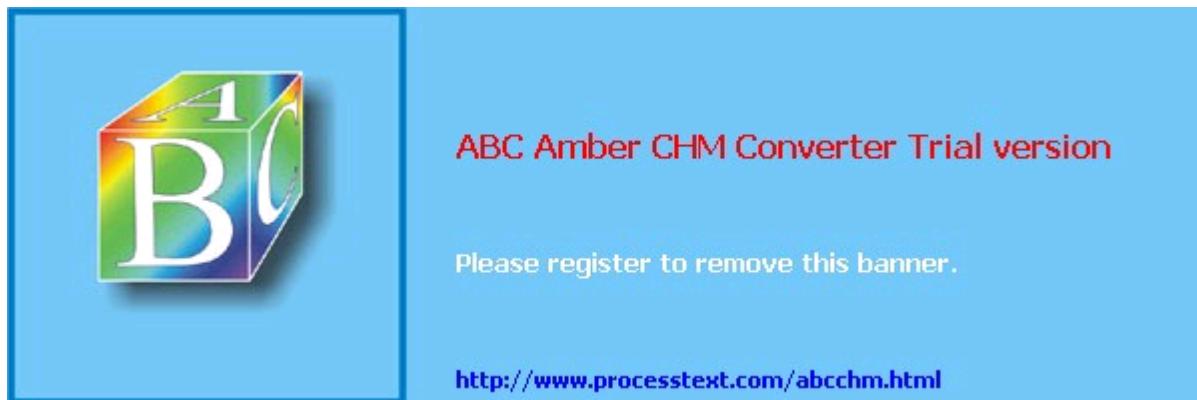
In the previous chapter, we upgraded the JAW Motors application by adding the ability to run a credit check on a customer using JMS messaging and an MDB. We didn't want to make the user wait a long time for the credit verification to complete, so we deferred this process to the background. As you'll recall, the `CreditCheckProcessor` MDB received the credit check message, invoked a simulated external credit verification service, and printed the result. But this wasn't completely satisfying because the user on the web site didn't know the final result of the credit verification process. In this chapter, we'll upgrade the MDB to use the JavaMail API for sending an email notification message to the user. Along the way we'll show how to deploy and configure JavaMail on JBoss.

JavaMail 1.2 is a J2EE API that enables Java programs to send and receive email messages. With JavaMail, you can send text or HTML messages, and you have the option to include Multipurpose Internet Mail Extensions (MIME) attachments. To keep things simple, we'll send text messages without attachments.

Let's start by briefly reviewing where we left off in the last chapter.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

8.1. Running a Credit Check

If you'll recall, we added a "Run Credit Check" link to the JAW Motors homepage as shown in [Figure 8-1](#).

Figure 8-1. JAW Motors homepage

JAW Motors

[View Inventory](#)
[Run Credit Check](#)

When the user clicks on the "Run Credit Check" link, the Controller routes them to the Run Credit Check page as depicted in [Figure 8-2](#).

Figure 8-2. JAW Motors Run Credit Check page

[\[Return to Main Page\]](#)

Name	<input type="text"/>
SSN	<input type="text"/>
Email	<input type="text"/>
<input type="button" value="Submit Credit Info"/> <input type="button" value="Reset"/>	

The user enters his name, Social Security Number (SSN), and email address in the form, and presses the "Submit Credit Info" button. When a customer requests a credit check, he won't have to wait while for the external credit verification process to complete. The Controller Servlet sends a JMS message asynchronously to back end components that process business logic for the request. The customer gets routed back to the JAW Motors home page and then is free to continue using the JAW Motors web site while the credit check completes in the background.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

8.2. Sending Email Messages with JavaMail

To upgrade the JAW Motors application to send email messages using JavaMail, we need to do the following:

- Add Java Code:
 - Call the JavaMail utility class from the MDB .
 - Write a utility class to encapsulate sending an email message with JavaMail.
 - Factor out JNDI calls into the `ServiceLocator`.
- Upgrade deployment:
 - Add JavaMail-based JNDI reference settings to the EJB-based deployment descriptors (`ejb-jar.xml` and `jboss.xml`) .
 - Automate JavaMail-based JNDI reference settings with XDoclet.
 - Configure JavaMail on JBoss with an MBean.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

8.3. Upgrading the MDB to Send an Email Message

If you'll recall from the JMS and MDB chapter, the `CreditCheckProcessor` MDB's `onMessage()` method consumes the message and invokes a simulated external credit verification service. [Example 8-1](#) shows the changes necessary to send an email message with JavaMail.

Example 8-1. CreditCheckProcessorBean.java

```
public class CreditCheckProcessorBean implements MessageDrivenBean,
MessageListener
{
    private static final String JAVAMAIL_SESSION =
            "java:comp/env/mail/JawJavaMailSession";

    private static final String JAW_MOTORS_EMAIL_ADDRESS =
            "credit.check@jbossatwork.com";

    private static final String CREDIT_VERIFICATION_RESULT = "Credit Check
Result";
    ...

    public void onMessage(Message message) {
        System.out.println(
                "CreditCheckProcessorBean.onMessage( ): Received message.");

        try {
            if (message instanceof ObjectMessage) {
                ObjectMessage objMessage = (ObjectMessage) message;
                Object obj = objMessage.getObject();

                if (obj instanceof CreditCheckRequestDTO) {
                    String result = null;
                    CreditCheckRequestDTO creditCheckReq =
                            (CreditCheckRequestDTO)
                    obj;

                    System.out.println("Verifying Credit ...");
                    result = CreditVerificationService.verifyCredit(
                            creditCheckReq);

                    System.out.println("Credit Check Result = [" + result +
                            "]");
                    sendNotificationEmail(creditCheckReq, result);
                } else {
                    System.err.println(
                            "Expecting CreditCheckRequestDTO in
Message");
                }
            } else {
                System.err.println("Expecting Object Message");
            }
        } catch (Throwable t) {
            t.printStackTrace();
        }
    }

    private void sendNotificationEmail(CreditCheckRequestDTO creditCheckReq,
                                      String result) {
        javax.mail.Session javaMailSession = null;

        try {
            javaMailSession =
                    ServiceLocator.getJavaMailSession(JAVAMAIL_SESSION);
        }
    }
}
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

8.4. Sending an Email Message

Creating and sending an email message with JavaMail requires the following steps:

- Get a JavaMail Session (which we did in the MDB's `sendEmailNotification()` method).
- Create the message.
- Set the "From" email address.
- Set the recipients' email addresses.
- Set the subject.
- Create the body and add it to the message.
- Send the message.

JavaMail requires a lot of low-level tedious API calls to send an email message, so we've encapsulated everything into the `TextEmail` utility object in [Example 8-2](#).

Example 8-2. TextEmail.java

```
package com.jbossatwork.util;

import javax.mail.*;
import javax.mail.internet.*;

import java.util.*;

/**
 * TextEmail defines utility methods for the JavaMail API, which provides
 * a platform independent and protocol independent framework to build Java
 * technology-based mail and messaging applications.
 */
public class TextEmail {
    ...

    private Session session;
    private InternetAddress sender = new InternetAddress( );
    private String subject = new String( );
    private StringBuffer body = new StringBuffer( );
    private List recipients = new ArrayList( );
    ...

    public TextEmail(Session session) throws EmailException {
        setSession(session);
    }

    ...

    public void setSession(Session session) {
        this.session = session;
    }

    ...

    public void send( ) throws EmailException {

        try {
            ...
            InternetAddress[ ] recipientsArr = (InternetAddress[ ])
                recipients.toArray(new InternetAddress[0]);
        }
    }
}
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

8.5. JavaMail-Based JNDI References in EJB Deployment Descriptors

In previous chapters we've used the `ejb-jar.xml` and `jboss.xml` file to describe and deploy EJBs and JNDI resources. Here are the new JavaMail -based JNDI references in `ejb-jar.xml` that enable the `CreditCheckProcessor` MDB to use a JavaMail Session, as in [Example 8-5](#).

Example 8-5. ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
           http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd" version="2.1">

  <enterprise-beans>
    ...
    <message-driven>
      <display-name>CreditCheckProcessorMDB</display-name>
      <ejb-name>CreditCheckProcessor</ejb-name>
      ...
      <resource-ref>
        <res-ref-name>mail/JawJavaMailSession</res-ref-name>
        <res-type>javax.mail.Session</res-type>
        <res-auth>Container</res-auth>
      </resource-ref>
      ...
    </message-driven>
    ...
  </enterprise-beans>
</ejb-jar>
```

The `<resource-ref>` elements specify the JNDI resources available to the `CreditCheckProcessorMDB` (or any POJO that it calls). `<res-ref-name>` is the JNDI name for the resource `java:comp/env/mail/JawJavaMailSession`. Notice that you don't have to specify `java:comp/env/` because it is the assumed prefix. The `<res-type>` for the `mail/JawJavaMailSession` is a JavaMail Session, and `javax.jms.Session` is its fully qualified class name. We want JBoss to manage our JavaMail resources, so we set `<res-auth>` to `Container`.

A JNDI resource is linked into an application only if we ask for it. JBoss binds resources under its in-JVM context, `java:/`. The `jboss.xml` file provides a mapping from the J2EE-style ENC names and the local JBoss-specific JNDI names that JBoss uses to deploy a JavaMail `Session`. [Example 8-6](#) shows the JavaMail-related JNDI references for the `CreditCheckProcessor` MDB in `jboss.xml`.

Example 8-6. jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"
           "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">

<jboss>
  <enterprise-beans>
    ...
    <message-driven>
      <ejb-name>CreditCheckProcessor</ejb-name>
      <destination-jndi-name>queue/CreditCheckQueue</destination-jndi-name>

      <resource-ref>
        <res-ref-name>mail/JawJavaMailSession</res-ref-name>
        <jndi-name>java:/Mail</jndi-name>
      </resource-ref>
    </message-driven>
  </enterprise-beans>
</jboss>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)

8.6. Automating JavaMail-Based JNDI References with XDoclet

As in previous chapters, we don't want to hardcode our deployment descriptors. We need to add XDoclet tags to the `CreditCheckProcessor` MDB so the Ant build process can add the JavaMail JNDI settings to the J2EE standard (`ejb-jar.xml`) and JBoss-specific (`jboss.xml`) EJB deployment descriptors in [Example 8-7](#).

Example 8-7. CreditCheckProcessorBean.java

```
/**  
 * ...  
  
 * @ejb.resource-ref  
 *   res-ref-name="mail/JawJavaMailSession"  
 *   res-type="javax.mail.Session"  
 *   res-auth="Container"  
  
 * @jboss.resource-ref  
 *   res-ref-name="mail/JawJavaMailSession"  
 *   jndi-name="java:/Mail"  
  
 */  
public class CreditCheckProcessorBean implements MessageDrivenBean,  
MessageListener  
{  
    ...  
}
```

The `@ejb.resource-ref` XDoclet tag generates the `<resource-ref>` element for the `JawJavaMailSession` in `ejb-jar.xml`, and the `@jboss.resource` XDoclet tag generates the corresponding `<resource-ref>` element in `jboss.xml`.

At this point, we've written code to send an email message with JavaMail and added JavaMail-based JNDI references to our EJB deployment descriptors. To complete the deployment, we'll deploy JavaMail on JBoss.

[◀ PREV](#)[NEXT ▶](#)

 PREV

NEXT 

8.7. Deploying JavaMail on JBoss

We now need to configure JBoss so we can use it as a JavaMail provider. JBoss manages a JavaMail Session as a JMX MBean that creates the Session and registers its JNDI name. The JBoss JavaMail XML descriptor, `$JBOSS_HOME/server/jbossatwork/deploy/mail-service.xml`, sets up a JMX MBean that configures a JNDI-based JavaMail Session in [Example 8-8](#).

Example 8-8. mail-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE server>

<server>

    <mbean code="org.jboss.mail.MailService"
           name="jboss:service=Mail">
        <attribute name="JNDIName">java:/Mail</attribute>
        <attribute name="User">yourUserId</attribute>
        <attribute name="Password">yourPassword</attribute>
        <attribute name="Configuration">
            <configuration>
                <!-- Set the protocol for your mail server -->
                <property name="mail.store.protocol" value="pop3"/>
                <property name="mail.transport.protocol" value="smtp"/>

                <!-- Configure the POP3 Server -->
                <property name="mail.pop3.host" value="yourIsp.pop3.host"/>

                <!-- Configure the SMTP gateway server -->
                <property name="mail.smtp.host" value="yourIsp.smtp.host" />
                <property name="mail.smtp.port" value="25"/>

                <property name="mail.debug" value="true"/>
            </configuration>
        </attribute>
    </mbean>

</server>
```

The `mail-service.xml` file configures a JavaMail Session with the email account properties that you use to connect with your email service provider:

- Mail Store Protocol
- Mail Transport Protocol
- SMTP server name
- POP server name
- An email account user ID
- An email account password

These are standard JavaMail properties, and you can find a complete listing of them in [Appendix A](#) of the JavaMail Design Specification at: <http://java.sun.com/products/javamail/JavaMail-1.2.pdf>.

The example above uses bogus values for its ISP settings, so you'll need to edit the `$JBOSS_HOME/server/jbossatwork/deploy/mail-service.xml` file and fill in the protocol and account settings you use to access your ISP's mail server. You could change the JNDI name to something other than `java:/Mail`, but every JBoss installation we've seen uses this value to deploy its JavaMail Session. Therefore we recommend sticking with the default value to be consistent with the way most people use JBoss.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

8.8. JavaMail Checklist

Before we move on to test the JAW Motors application's new email functionality, let's recap what we've done to implement JavaMail:

- Added Java Code:
 - Called the `TextEmail` JavaMail utility class from the `CreditCheckProcessor` MDB
 - Wrote a `TextEmail` utility class to encapsulate sending an email message with JavaMail
 - Factored out JNDI calls into the `ServiceLocator`
- Upgraded deployment:
 - Added JavaMail-based JNDI reference settings to the EJB-based deployment descriptors (`ejb-jar.xml` and `jboss.xml`)
 - Automated JavaMail-based JNDI reference settings in the `CreditCheckProcessor` MDB with XDoclet
 - Configured JavaMail on JBoss with an MBean

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

8.9. Testing the Credit Check Notification Email

Now that we've developed and deployed JavaMail code to send an email notification message to the user, let's test our application to ensure that everything still works properly. Here are the steps to build and deploy the application:

- Type `ant` in the root directory of `ch08` to build the project.
- Shut down JBoss so the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBoss_HOME/server/default/deploy` directory.
- Start JBoss back up.
- Visit <http://localhost:8080/jaw> in a web browser.

Click on the "Run Credit Check" link on the JAW Motors home page, enter data on the "Run Credit Check" form, and press the "Submit Credit Info" button. You should be routed back to the main page, and in a minute or two you should see a message that looks like this in your email client's Inbox:

From: credit.check@jbossatwork.com
To: yourName@host.domain
Subject: Credit Check Result

Pass Credit Check

The credit check randomly passes or fails, so you could also see "Fail Credit Check" in the email message body.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

8.10. Looking Ahead ...

This chapter covered JavaMail. We upgraded the JAW Motors application by using JavaMail to send an email notification to the customer after running a credit check. Along the way, we showed how to deploy and configure JavaMail on JBoss.

For our purposes, we're now finished with core functionality for the JAW Motors application. In the next chapter, we'll show how to secure the application.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

Chapter 9. Security

If you've worked through all the previous chapters, you have a fully functional vertical slice of the JAW Motors application that allows you to run a credit check and view, add, edit (update), delete, and buy cars. Although this works, there's a gaping hole anyone with a browser who knows the application's URL can modify JAW Motors' inventory. So we need to add security to the application. In this chapter, we'll secure the "Car Inventory" and "Add/Edit Car" pages so that only authorized users can modify cars in the inventory. We won't secure the "Buy Car" or "Run Credit Check" pages (and their underlying functionality) because we still want all users to be able to buy a car or run a credit check without having to log in. We'll discuss J2EE web-based security, Java Authentication & Authorization Service (JAAS), and EJB security. Along the way we'll show how to deploy these security mechanisms on JBoss.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

9.1. J2EE Security

Security is an important part of J2EE application architecture because the J2EE components and tiers used in a system's architecture determine the choice of security technologies. If an application uses only web-based technologies, then it only needs to restrict access to JSPs, Servlets, and so on. But EJBs are now part of the JAW Motors architecture, so they must be protected as well. The system must create a security context that encompasses the entire J2EE stack from frontend web pages to backend business logic and data. We need a unified security mechanism that propagates the user's credentials to all components in the application.

The two fundamental concepts in J2EE security are:

Authentication

Answers the following questions:

- Who is attempting to access the system?
- Is this person allowed to access the system?

Authorization

Determines what an authenticated user can access in an application.

Authentication is an important aspect of a J2EE application's architecture and security strategy, and ensures that only valid users or entities can use the system's resources. Authentication is the front line of defense in protecting sensitive business logic and data from users. Authentication identifies a user in the system, and requires the user to log on just as they would log on to an operating system or database. Users identify themselves to the system by supplying credentials, which could be in the form of passwords, certificates, or keys. If the user enters a valid username and password, the user can access sensitive portions of the web site; otherwise, access is denied.

Although restricting access on internal business functions and web pages to known users of the JAW Motors is a good first step in securing the system, it still isn't enough. We know who the user is, but what can they do in the system? What are they not allowed to do? How can we ensure that users see only what they're allowed to access? Authorization answers these questions and strengthens security by adding the concept of roles to our security realm. Each role represents different types of users, and the JAW Motors application has the following roles:

Manager

A manager is an administrative user who can modify the JAW Motors inventory.

Guest

A guest is a user who can only view the JAW Motors inventory.

Although there is no need to protect public pages and their underlying business logic, we must prevent unauthenticated/unauthorized users from accessing protected web pages and business functions.

Let's start by securing the web tier and working our way down through the architecture.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

9.2. Web-Based Security

To secure the web site, we need to do the following:

- Protect the administrative pages:
 - Restrict access based on the URL pattern.
 - Associate security roles with the URL.
 - Create security roles for the JAW Motors application.
- Choose an Authentication mechanism and implement it.
- Automate extra `web.xml` settings with XDoclet.
- Create a security realm that associates a user with the roles he plays in the system.
- Configure a JAAS `LoginModule` that's tied to the security realm.
- Deploy the JAAS-based security realm with the JBoss container.
- Protect MVC administrative actions:
- Restrict access based on the URL pattern.
- Propagate the correct user credentials from the web tier:
 - Establish a default user identity for non-secure web access.
 - Use the right user identity for secure web access.

9.2.1. Protecting the Administrative Pages

J2EE provides Declarative Security, so rather than writing code to protect our resources, we can accomplish this through URL patterns and deployment descriptors. If you'll recall, the Car Inventory page (`carList.jsp`), as shown in [Figure 9-1](#), enables you to view and modify the JAW Motors inventory.

Figure 9-1. JAW Motors Car Inventory page

[[Add Car](#)]

Delete	Action	Make	Model	Model Year	Buy Car
<input type="checkbox"/>	Edit	Toyota	Camry	2005	Buy
<input type="checkbox"/>	Edit	Toyota	Corolla	1999	Buy
<input type="checkbox"/>	Edit	Ford	Explorer	2005	Buy

[Delete Checked](#) [Reset](#)

We also must protect the Add/Edit Car page (`carForm.jsp`) you see this page when you press the "Add Car" or "Edit" link on the Car Inventory page) ([Figure 9-2](#)).

Figure 9-2. JAW Motors Add/Edit Car page

[[Return to List](#)]

Make	<input type="text"/>
Model	<input type="text"/>
Model Year	<input type="text"/>
Save	Reset

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

9.3. Restricting Access with web.xml

We restrict access to the administrative page URLs in `web.xml` as in [Example 9-1](#).

Example 9-1. web.xml

```

...
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            JAW Application protected Admin pages.
        </web-resource-name>
        <description>Require users to authenticate.</description>
        <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <description>
            Allow Manager role to access Admin pages.
        </description>
        <role-name>Manager</role-name>
    </auth-constraint>

</security-constraint>

<security-role>
    <description>JAW Managers</description>
    <role-name>Manager</role-name>
</security-role>
...

```

The `<security-constraint>` element protects the administrative pages by specifying:

- A protected URL pattern for the admin pages, `/admin/*` the protected JSPs reside in the web application's `/admin` directory.
- That only users who are in the `Manager` role can access the administrative page. This role must be specified in a separate `<security-role>` element.

To be complete, we've also modified the Controller Servlet in [Example 9-2](#) to prefix all administrative pages with the `admin/` URL.

Example 9-2. ControllerServlet.java

```

public class ControllerServlet extends HttpServlet
{
    ...
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        ...
        // perform action
        ...
        else if(MODIFY_CAR_LIST_ACTION.equals(actionName))
        {
            ...
            destinationPage = "/admin/carList.jsp";
        }
        ...
    }
}

```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

9.4. JAAS

The Java Authentication & Authorization Service (JAAS) enables an application to protect its resources by restricting access to only those users with proper credentials and permissions. JAAS provides a layer of abstraction between an application and its underlying security mechanisms, making it easier to change security technologies and realms without impacting the rest of the system. JAAS is a standard Java extension in J2SE 1.4, and provides pluggable authentication to give application designers a wide choice of security realms:

- DBMS
- Application Server
- LDAP
- Operating System (UNIX or Windows NT/2000)
- File System
- JNDI
- Biometrics

JAAS supports single sign-on for an application. Rather than forcing the user to log in to a web site, and then log in again to a forum or a backend legacy system used by the application, JAAS wraps all of this in one central login event to make it easier to coordinate access to all systems that the user needs. We chose JAAS as the basis for our security strategy because:

- It provides a security context that covers the entire J2EE architecture from the web tier to the EJB tier.
- It is application server neutral.
- It integrates with the Java 2 security model.
- It is part of the J2SE 1.4 extension API.
- It is more sophisticated than the other authentication mechanisms and provides more functionality.
- It supports single sign-on by coordinating multiple security realms.
- It addresses authorization in addition to authentication.
- It provides good encapsulation for authentication and authorization, enabling an application to be independent of the underlying security mechanisms used.
- JBoss bases its security mechanism on JAAS.

Although this isn't a JAAS book, we've added more detailed information on JAAS in [Appendix C](#) JAAS Tutorial.

9.4.1. LoginModule

The `LoginModule` logs a user/Subject into a security realm based on their username and password. A `LoginModule` could interact with an operating system, a database, JNDI, LDAP, or a biometric device like a retinal scanner or touch pad. Application developers normally don't need to know very much about `LoginModule`s because the `LoginContext` invokes them on behalf of an application. So your code never interacts with `LoginModule`s. To add or remove a `LoginModule` used by your application, you need to modify only the `LoginModule` Configuration file—your code remains unchanged. This indirection enables an application to be independent of the underlying security mechanisms used.

Although you could write your own `LoginModule`, it is usually unnecessary because of the abundance of quality third-party Open Source implementations available. You only need to know how to configure (in the `LoginModule` Configuration file) and deploy them for your

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

9.5. Deploying a JAAS-Based Security Realm on JBoss

There is no standard for integrating JAAS deployment with a J2EE application server, so each server has its own way to set up `LoginModule` configuration and domain names. So, we need to:

- Configure the `LoginModule`.
- Add the `LoginModule`'s security domain name to `jboss-web.xml`.

9.5.1. JBoss LoginModule Configuration

To Configure a `LoginModule` in JBoss, you have a couple of options.

- Add the `LoginModule` configuration data to the JBoss default `$JBOSS_DIST/server/default/conf/login-config.xml`.
- Create your own custom `LoginModule` configuration file in `$JBOSS_DIST/server/default/conf` that has the same structure as `login-config.xml` and conforms to the `security_config.dtd`.

9.5.2. Custom LoginModule Configuration

We chose to create our own separate `LoginModule` Configuration file, `jaw-login-config.xml`, because we didn't want to co-mingle our application-specific JAAS `LoginModule` configuration with JBoss-internal `LoginModule` settings. Co-mingling `LoginModule` settings is bad because each time you upgrade to a new version of JBoss, you have to re-add your `<application-policy>` elements to the default `login-config.xml` file to make things work again. To deploy the `LoginModule` configuration file to JBoss, the Ant build script copies the `jaw-login-config.xml` file from the `ch09-a/src/META-INF` directory to the JBoss configuration directory `$JBOSS_HOME/server/default/conf`. The `jaw-login-config.xml` file in [Example 9-5](#) looks just like the JBoss default `login-config.xml`, but our file contains only application-specific `<application-policy>` elements.

Example 9-5. jaw-login-config.xml

```

<?xml version='1.0'?>
<!DOCTYPE policy PUBLIC
  "-//JBoss//DTD JBOSS Security Config 3.0//EN"
  "http://www.jboss.org/j2ee/dtd/security_config.dtd">
<policy>

  <application-policy name = "JawJaasDbRealm">
    <authentication>
      <login-module code =
        "org.jboss.security.auth.spi.DatabaseServerLoginModule"
        flag = "required">
        <module-option name="unauthenticatedIdentity">guest</module-option>
        <module-option name="password-stacking">useFirstPass</module-option>
        <module-option name="dsJndiName">java:/JBossAtWorkDS</module-option>
        <module-option name="principalsQuery">SELECT PASSWORD FROM USER WHERE
NAME=?</module-option>
        <module-option name="rolesQuery">SELECT ROLE.NAME, 'Roles' FROM ROLE,
USER_ROLE, USER WHERE USER.NAME=? AND USER.ID=USER_ROLE.USER_ID AND ROLE.ID =
USER_ROLE.USER_ID</module-option>
      </login-module>
    </authentication>
  </application-policy>

</policy>

```

JBoss uses an MBean that reads the `$JBOSS_HOME/server/default/conf/jaw-login-config.xml` file at startup time to configure its security domains. Each `<application-policy>` element

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

9.6. Testing Secure JSPs

We've taken a lot of steps to get here, but we're now ready to take our new secure web site for a test drive. First, let's try to access one of the protected JSPs directly. Here are the steps to build and deploy the application:

- Type `ant` in the root directory of `ch09-a` to build the project.
- Shut down JBoss so the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBOSS_HOME/server/default/deploy` directory. The Ant build script also deploys:
 - The MBean service file (`jaw-login-config-service.xml`, which tells JBoss that we're using our own LoginModule Configuration file) to the `$JBOSS_HOME/server/default/deploy` directory.
 - The LoginModule Configuration file (`jaw-login-config.xml`) to the `$JBOSS_HOME/server/default/conf` directory.
- Start JBoss back up.
- Go to the `ch09-a/sql` sub-directory and type `ant` to modify the database.
- Visit <http://localhost:8080/jaw/admin/carList.jsp> in a web browser.

The Servlet container should re-direct you to the login page and you'll see the login page, `/login.jsp`, as shown in [Figure 9-3](#).

Figure 9-3. JAW Motors Login page

User Name:	<input type="text"/>
Password:	<input type="password"/>
<input type="button" value="Save"/>	<input type="button" value="Reset"/>

When the user presses the "Save" button, the browser sends the user ID and password to the Servlet Container, which validates the user's credentials against a security realm. If the user logs in successfully, the Servlet container takes him to the "Car Inventory" page. You'll notice that no cars are displayed because we bypassed the Controller Servlet that pulls the cars from the database before rendering the page. Otherwise, the container sends the user to the login error page, `/loginError.jsp` as depicted in [Figure 9-4](#).

Figure 9-4. JAW Motors Login Error page

A Login error occurred

[Try again](#)

[Back to main page](#)

The user can either return to the login page to try another user I and password or go back to the JAW Motors home page.

We have now successfully locked down the administrative pages, but this isn't good enough. Now exit the browser (to end your session) and re-start your browser. Try to use the JAW Motors web site by visiting the home page: <http://localhost:8080/jaw>, as shown in [Figure 9-5](#).

Figure 9-5. JAW Motors Home page

JAW Motors

[Modify Inventory](#)

[View Inventory](#)

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

9.7. Protecting the Administrative Actions

We can continue to use J2EE Declarative Security, as shown earlier, to protect the administrative actions like `modifyCarList`, `addCar`. To lock down the administrative actions, we'll take steps similar to those we used to protect the JSPs:

- Associate the administrative actions with an URL pattern.
- Protect the new administrative action URL pattern.

To associate the admin actions with an URL pattern, we prefix them with `admin/` in the JSPs and in the Controller Servlet. For example, `index.jsp` (the main page) now invokes the `modifyCarList` action through the Controller Servlet, as in [Example 9-9](#).

Example 9-9. index.jsp

```
<html>
  <head>
    ...
  </head>
  <body>
    <h1>JAW Motors</h1>
    <a href="controller/admin/modifyCarList">Modify Inventory</a><br/>
    <a href="controller/viewCarList">View Inventory</a><br/>
    <a href="controller/viewCreditCheckForm">Run Credit Check</a>
  </body>
</html>
```

The `viewCarList` action takes you to a read-only page, so it doesn't need any extra security. To fully protect the administrative action URLs, we add a new `<url-pattern>` to `web.xml` so that it now looks like [Example 9-10](#).

Example 9-10. web.xml

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      JAW Application protected Admin pages and actions.
    </web-resource-name>
    <description>Require users to authenticate.</description>
    <url-pattern>/admin/*</url-pattern>
    <url-pattern>/controller/admin/*</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <description>
      Allow Manager role to access Admin pages.
    </description>
    <role-name>Manager</role-name>
  </auth-constraint>

</security-constraint>
<security-role>
  <description>JAW Managers</description>
  <role-name>Manager</role-name>
</security-role>

  ...
```

The new `/controller/admin/* <url-pattern>` (a sub-element of `<web-resource-collection>`) finally gives us what we want: it forces the user to log in before accessing secure actions. As before, we add this new `<url-pattern>` element to the `webapp/xdoclet/merge/web-security.xml` file, and XDoclet adds the new security to `web.xml` on our behalf.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

9.8. Web Security Checklist

Before we move on to securing the EJB tier, let's recap what we've done so far:

- Protected the administrative pages by:
 - Restricting access based on the `/admin/*` URL pattern in `web.xml`
 - Associating security roles with the `/admin/*` URL pattern in `web.xml`
 - Moving the administrative pages beneath the `/admin` sub-directory in `WEB-INF`
 - Creating security roles for the JAW Motors application in `web.xml`
- Implemented FORM-based Authentication by:
 - Adding a `<login-config>` element to `web.xml` and tying it to a security realm
 - Creating a login page, `login.jsp` with a form that follows FORM-based Authentication naming conventions
 - Developing a login error page `loginerror.jsp`
- Automated extra `web.xml` settings with the `servlets.xml`, `servlet-mappings.xml`, and `web-security.xml` XDoclet merge files
- Created a security realm in the JAW Motors database that associates a user with the roles they play in the system
- Deployed the JAAS-based security realm with the JBoss container by:
 - Configuring a JAAS `LoginModule` that's tied to the database security realm using `$JBOSS_HOME/server/default/conf/$JBOSS_HOME/server/default/conf` and `$JBOSS_HOME/server/default/deploy/jaw-login-config-service.xml`
 - Adding the JAAS domain settings to `jboss-web.xml`
- Added a read-only page and MVC action to ensure that we can still access non-secure resources without logging in
- Protected MV administrative actions by:
 - Modifying JSPs and the Controller Servlet to prefix all administrative action URLs with `/admin/`
 - Modifying `web.xml` with the new `/controller/admin/* <url-pattern>` element to lock down the administrative action URLs
- Propagated the correct user credentials from the web tier:
Established a default `guest` user identity for non-secure actions and pages in `web.xml`
- Used the `Manager` identity for secure actions and pages in `web.xml`



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

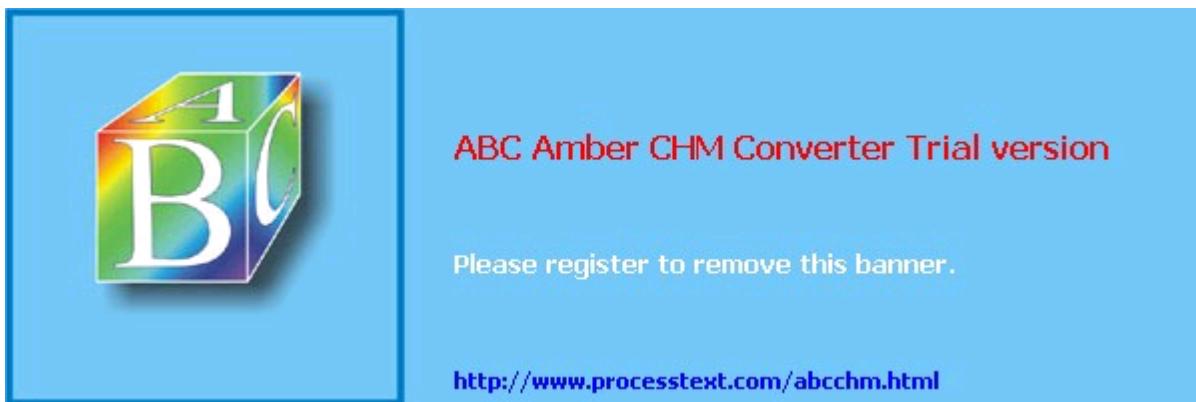
NEXT ▶

9.9. Integrating Web Tier and EJB Tier Security

It's taken a while to get here, but now that we've secured the web tier, we have the core infrastructure in place to secure the rest of the JAW Motors application. Although we've protected access to the `InventoryFacadeBean` EJB through the Controller Servlet in the web application, the EJB is still vulnerable. Unauthenticated/unauthorized external applications could look up the `InventoryFacadeBean` and access its administrative methods `saveCars()` and `deleteCars()`. We must protect the EJB tier by securing the administrative methods on the `InventoryFacadeBean`, yet still allow non-secure access to the non-administrative methods `listAvailableCars()`, `findCar()`, and `buyCar()`. We'll show how the JBoss security manager, in keeping with the J2EE specification, propagates the user's credentials from the web tier to the EJB container. We now discuss EJB security in greater detail.

◀ PREV

NEXT ▶



 PREV

NEXT 

9.10. EJB Security

To secure the EJB tier, we need to do the following:

- Deploy the JAAS-based security realm with the JBoss container.
- Protect the EJB:
 - Allow access to non-secure methods.
 - Configure access to administrative methods.
 - Add security roles.
- Automate extra `ejb-jar.xml` settings with XDoclet.

9.10.1. JAAS Domain in `jboss.xml`

The security domain in `jboss.xml` defines a security domain used by all EJBs in the application. To join the global security domain for the entire JAW Motors application, the security domain must match the "JawJaasDbRealm" JAAS application name from `login-config.xml` and the `<security-domain>` element in `jboss-web.xml`. The `<security-domain>` element defines a single security domain used by all EJBs in the application. The `<security-domain>` element comes before the elements that define the JNDI-based resources. [Example 9-13](#) shows the `<security-domain>` element in `jboss.xml`.

Example 9-13. `jboss.xml`

```
<jboss>
  <security-domain>java:/jaas/JawJaasDbRealm</security-domain>
  ...
</jboss>
```

The `<security-domain>` uses `java:/jaas/JawJaasDbRealm` because it is the JBoss-specific JNDI name used when JBoss deploys the `LoginModule` as a managed service. The pattern here is that JBoss prefixes its JAAS JNDI names with `java:/jaas`.

9.10.2. Automating JAAS Domain Settings in `jboss.xml`

If you recall from the Session Bean chapter, we used XDoclet's Ant `<ejbdoclet>` task and its `<jboss>` subtask to generate the J2EE standard `ejb-jar.xml` and `jboss.xml` JBoss-specific EJB deployment descriptors, respectively. We now add a `securitydomain` attribute to the `<jboss>` subtask in the `ejb` sub-project's `build.xml` ([Example 9-14](#)) to generate the `<security-domain>` element in `jboss.xml`.

Example 9-14. `ejb/build.xml`

```
...
<ejbdoclet>
  ...
  <jboss version="4.0" destdir="${gen.source.dir}"
        securitydomain="java:/jaas/JawJaasDbRealm"/>
</ejbdoclet>
...
```

9.10.3. Protecting EJBs with `ejb-jar.xml`

J2EE provides Declarative Security, so we modify `ejb-jar.xml` in [Example 9-15](#) to configure EJB security.

Example 9-15. `ejb-jar.xml`

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

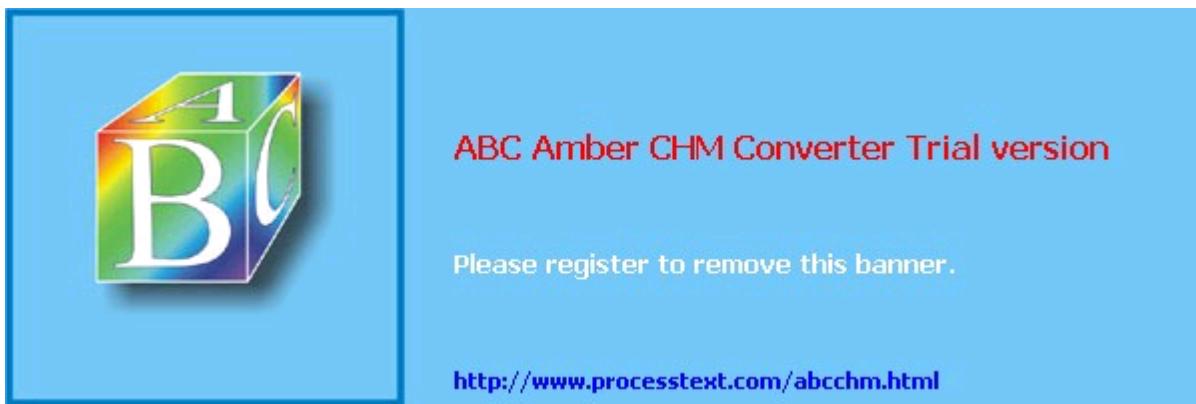
9.11. EJB Security Checklist

To secure the EJB tier, we did the following:

- Deployed the JAAS-based security realm with the JBoss container.
- Protected the EJB in `ejb-jar.xml`:
 - Added security roles.
 - Allowed callers with the unauthenticated `guest` or authorized `Manager` role to access non-secure methods.
 - Restricted access to administrative methods to users in the `Manager` role.
- Automated extra `ejb-jar.xml` settings with XDoclet.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

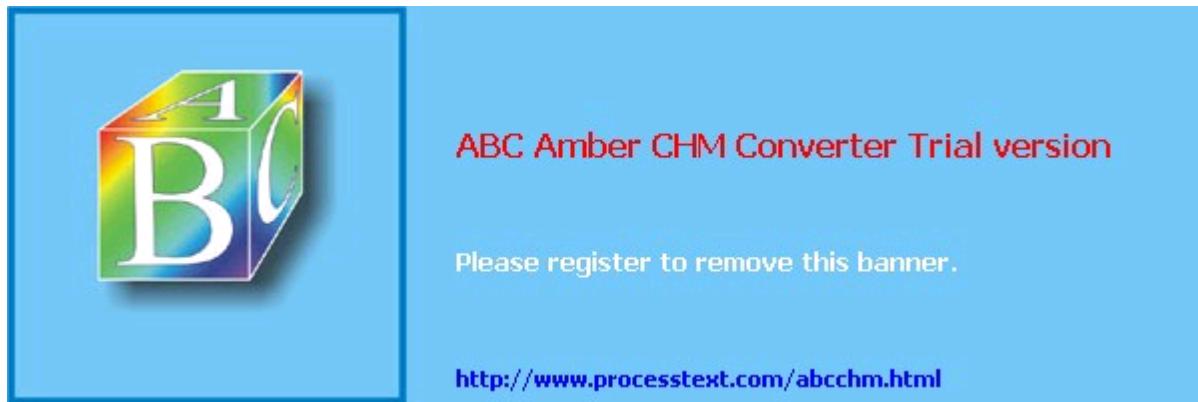
9.12. Looking Ahead ...

In this chapter, we secured the "Car Inventory" and "Add/Edit Car" pages so that only authorized users can modify cars in the inventory. We discussed J2EE web-based security, JAAS, and EJB security. Along the way, we showed how to deploy these security mechanisms on JBoss.

We've developed and secured the JAW Motors application. In the next and final chapter, we'll show how to expose a portion of the application as a Web service.

[◀ PREV](#)

[NEXT ▶](#)



◀ PREV

NEXT ▶

Chapter 10. Web Services

JAW Motors has a fully functional, secure J2EE-based web site and the business is doing well. Harry Schmidlap, the company President, now wants to expand JAW Motors' business beyond the web site and boost sales by displaying its inventory on other related web sites. Another company, Virtual Big Auto Dealership (VBAD), has a high-traffic web site that consolidates the inventory of many auto dealerships. Thousands of customers use VBAD's service to find and purchase cars. Mr. Schmidlap views VBAD as an ideal trading partner due to the sheer volume of potential new customers they could bring to JAW Motors.

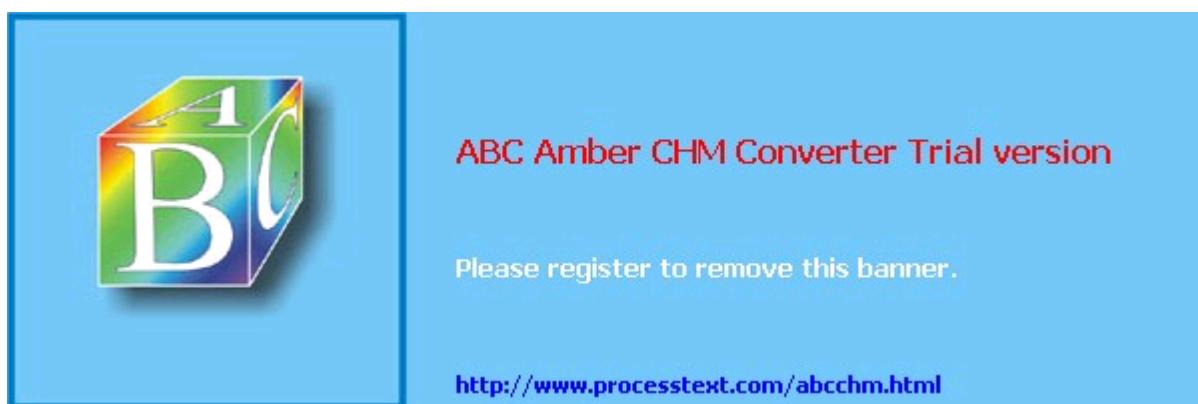
One problem currently prevents JAW Motors from sharing its inventory VBAD doesn't use J2EE. So none of the technologies we've shown so far will enable JAW Motors and VBAD to communicate. But VBAD has an experienced IS staff and knows how to use Web Services. Mr. Schmidlap has instructed Gunther Toady (with apologies to the restaurant chain and the cast of "Car 54, Where are You?"), the JAW Motors CTO, to look into Web Services and report back to the Board of Directors within two weeks with his results and findings.

This chapter shows how to deploy a portion of the JAW Motors application as a Web Service so it can work with non-Java clients. We'll show how to expose an EJB as a Web Service by using XDoclet and Java Web Services Developer Pack (JWSDP) to deploy it on JBoss. We'll finish by writing an Axis client that uses/consumes our Web Service.

Although we're going to show all deployment descriptors, including the WSDL, we're not covering them in any depth because our focus is on how to deploy a J2EE-based Web Service. We recommend J2EE Web Services by Richard Monson-Haefel, if you want to know the gory details of WSDL and you'd like a detailed description of all the elements in the Web Service deployment descriptors.

◀ PREV

NEXT ▶



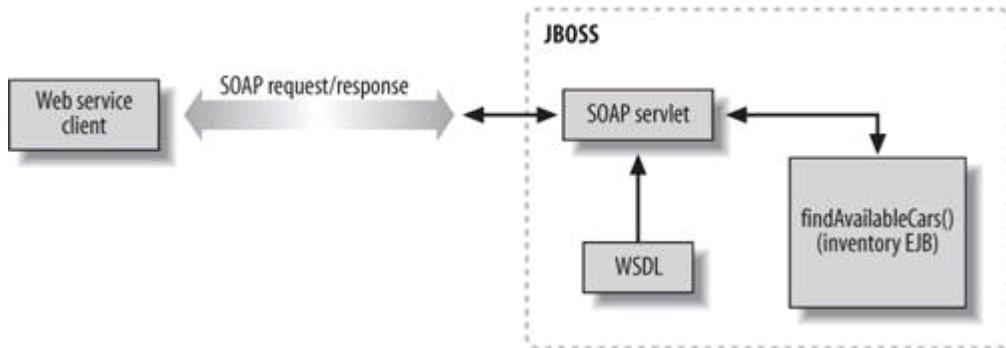
 PREV

NEXT 

10.1. Web Services Architecture

[Figure 10-1](#) shows how the VBAD client uses Web Services to access the JAW Motors inventory.

Figure 10-1. Web Service invocation



VBAD's web site acts as an external client that invokes the JAW Motors `findAvailableCars()` Web Service to get information about all JAW Motors cars. Underneath the covers, the client uses a Web Service proxy object to marshal the method `findAvailableCars()` call as a Simple Object Access Protocol (SOAP) Request and sends it to the JBoss application server. The SOAP Servlet picks up the SOAP Request, looks up the `findAvailableCars` service in the Web Services Definition Language (WSDL) file and invokes the `findAvailableCars()` method on behalf of the client. The `findAvailableCars()` method finds all unsold cars in the JAW Motors database, packages them into `CarDTO` objects, and returns control to the SOAP Servlet. The SOAP Servlet then marshals the `CarDTO` objects into a SOAP Response and returns it to the caller. On the client side, the Web Service proxy object unmarshals the SOAP Response into `CarDTO` objects to be displayed on the VBAD web site.

Web Services has its own terminology, so let's wade through the alphabet soup:

SOAP

SOAP is an XML-based, platform-neutral, wire protocol that enables remote communication. Client and server communicate using SOAP Messages that contain a Header and a Body. The Header has routing information and the Body holds the request/response data.

WSDL

WSDL is an XML-based interface descriptor that describes a web service interface along with its parameters. WSDL registers your Web Service with your server in the same way `web.xml` registers Servlets and `ejb-jar.xml` registers EJBs.

Web Service Proxy

A Web Service Proxy is a set of objects that work together to encapsulate low-level SOAP communication details and invokes a Web Service on behalf of a client. The client just uses the Web Service proxy and is oblivious to the low-level network API. SOAP toolkits are available for most programming languages that use the WSDL (for the Web Service) to generate Web Service proxy code for that language.

SOAP Servlet

As of J2EE 1.4, most application servers use a Servlet to listen for SOAP Requests, route them to a Web Service, and return the result(s) as a SOAP Response.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

10.2. JBoss 4.x and Web Services

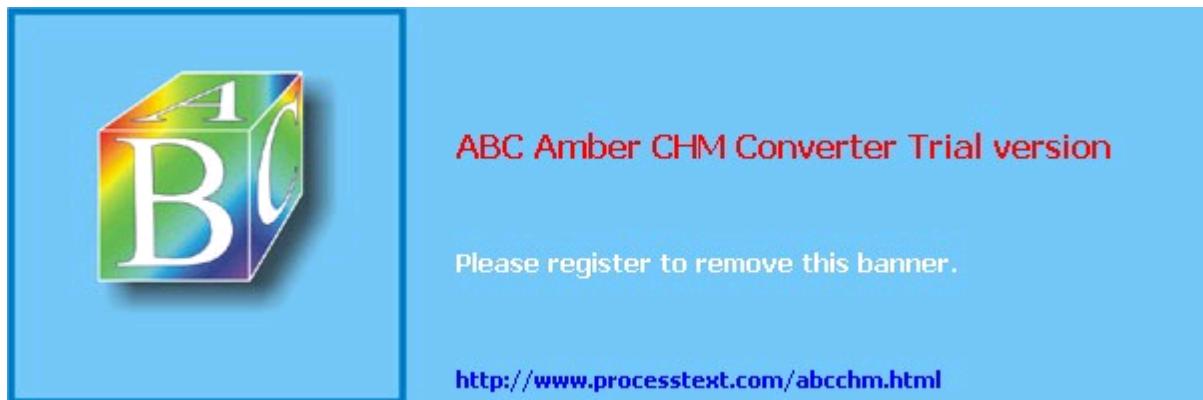
In previous versions of JBoss (pre 4.x), developers deployed Web Services by using a JBoss-specific tool called JBoss.NET (not to be confused with Microsoft .NET). Although JBoss.NET worked well and provided a highly automated way to create Web Services on JBoss, it was proprietary technology. In JBoss 4.x, JBoss.NET is deprecated in favor of JBossWS, JBoss's new J2EE 1.4-compliant Web Service implementation. JBossWS is based on Apache Axis (<http://ws.apache.org/axis>) and uses J2EE standard deployment descriptors and technologies.

JBoss and Web Services Issues

Bugs are a fact of life, and even though JBoss is an excellent product, JBoss 4.0.0 and 4.0.1 have problems deploying Web Services that use Custom Data types. Specifically, these versions of JBoss can't find serializers that convert custom data types between Java and WSDL. So you can't access Web Services that use Custom Data types with JBoss 4.0.0 and 4.0.1. This problem is fixed in JBoss 4.0.1sp1, so as long you use JBoss 4.0.1sp1 or later, everything will work properly.

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

10.3. J2EE 1.4 and Web Services

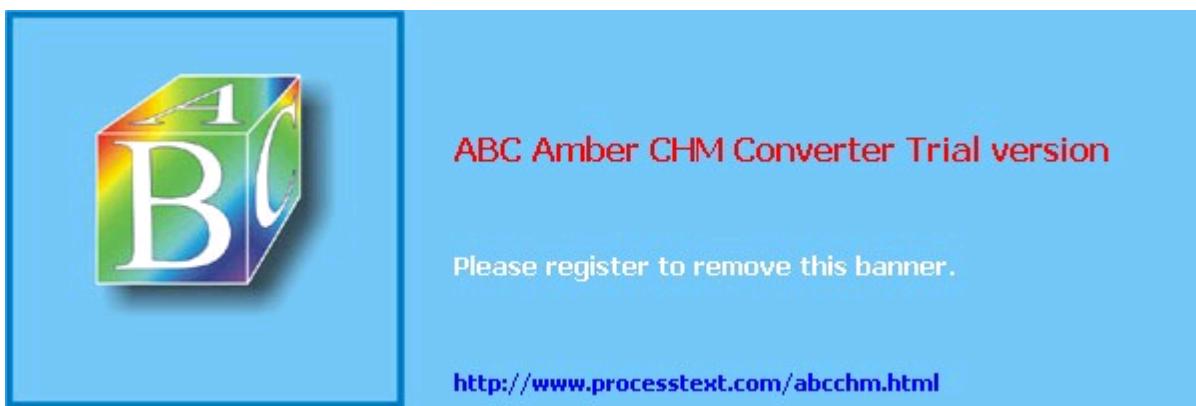
The main purpose of the J2EE 1.4 specification was to standardize Web Service deployment so Web Services would be portable and interoperable. With J2EE 1.4, you now use a set of standard deployment descriptors to deploy a Web Service so it can be deployed on any J2EE-compliant application server. The J2EE 1.4 specification mandates interoperability with other platforms, so a Web Service deployed on a J2EE application server will work and be compatible with non-Java clients written in: C#, Python, Perl, C++, and so on.

We'll introduce the new J2EE-Web Service deployment descriptors as we go through the process of deploying JAW Motors Inventory-related Web Services. On the server-side, we have to create a Service Endpoint Interface and register it with the deployment descriptors. In J2EE 1.4, you can implement a Service Endpoint as a Servlet or POJO deployed in a WAR file, or as a Stateless Session Bean, but we'll limit our discussion to EJB Service Endpoints.

Let's start by working our way down through the architecture.

◀ PREV

NEXT ▶



[◀ PREV](#)[NEXT ▶](#)

10.4. Implementing J2EE 1.4 Web Services

To expose the `InventoryFacadeBean`'s `findAvailableCars()` method as a J2EE 1.4-compliant Web Service, we need to do the following:

- Create a Service Endpoint Interface (SEI) to expose EJB methods as Web Service operations.
- Add a `<service-endpoint>` element to `ejb-jar.xml` to tell JBoss that the `InventoryFacadeBean` EJB implements the `InventoryEndpoint` Service Endpoint Interface.
- Generate the `webservices.xml` to register the Web Service and tie the `InventoryEndpoint` Service Endpoint Interface to an implementation the `InventoryFacadeBean`.
- Create the JAX-RPC Mapping File to define JAX-RPC type mappings for the parameters and return values for the `InventoryEndpoint`'s methods.
- Generate the WSDL to define the Web Service and tie it to XML Schema data types.
- Set the Web Service URL by modifying `jboss.xml` with the Inventory Web Service URL.
- Modify the `InventoryFacadeBean`:
 - Add Web Services-related XDoclet tags.
 - Add the `findAvailableCars()` method and use an XML Schema-compatible data type.%
- Upgrade Deployment by modifying the Ant build script:
 - Fix XDoclet-generated descriptors.
 - Modify the Web Service URL in `jboss.xml`.
 - Use JWSDP to generate the JAX-RPC and WSDL files.

As you'll see in the upcoming sections, the extra interface and descriptors required to deploy a Web Service are tedious and would be almost impossible to develop by hand. But don't lose heartwe can automate Web Service deployment with a combination of Ant, XDoclet, and JWSDP. However, before we show our new deployment process, it's important to know what we're automatinglet's start with the Service Endpoint Interface.

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)

10.5. Service Endpoint Interface (SEI)

A Service Endpoint Interface exposes business methods as Web Services that can be accessed by external clients. [Example 10-1](#) shows the `InventoryService` interface.

Example 10-1. InventoryEndpoint.java

```
package com.jbossatwork.ws;

/**
 * Service endpoint interface for InventoryFacade.
 */
public interface InventoryEndpoint
    extends java.rmi.Remote
{
    public com.jbossatwork.ws.CarDTOArray findAvailableCars( )
        throws java.rmi.RemoteException;
}
```

A Service Endpoint Interface acts as a server-side stub that shows your business methods to clients and serves the same purpose for a Web Service that an EJB Remote Interface does for an EJB. The `InventoryEndpoint` interface is a `Remote` interface, and the `findAvailableCars()` method throws a `RemoteException`. The `CarDTOArray` (that holds an array of `CarDTO` objects) returned by the `findAvailableCars()` method may look odd to you see the "[Web Services and Collections](#)" section for more information.

[◀ PREV](#)[NEXT ▶](#)

◀ PREV

NEXT ▶

10.6. Modifying ejb-jar.xml

Although we're already generating the J2EE-standard `ejb-jar.xml` descriptor to deploy EJBs, we have to add a `<service-endpoint>` element to [Example 10-2](#) to tell JBoss that the `InventoryFacadeBean` EJB implements the `InventoryEndpoint` interface.

Example 10-2. ejb-jar.xml

```
<enterprise-beans>
  ...
  <session>
    ...
      <display-name>InventoryFacadeSB</display-name>

      <ejb-name>InventoryFacade</ejb-name>
      ...

<service-endpoint>com.jbossatwork.ws.InventoryEndpoint</service-endpoint>

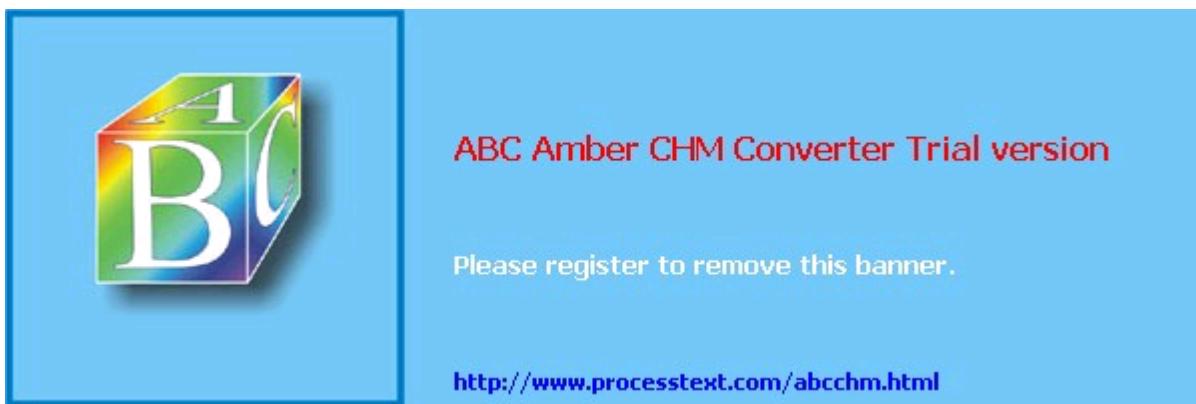
  ...
</session>
  ...

</enterprise-beans>
```

Now that we've created a Web Service Endpoint, we have to register it with JBoss.

◀ PREV

NEXT ▶



[◀ PREV](#)[NEXT ▶](#)

10.7. webservices.xml

The J2EE-standard `webservices.xml` file defines and registers the `InventoryService` Web Service, and ties the Service Endpoint Interface class (`com.jbossatwork.ws.InventoryEndpoint`) to the `InventoryFacadeBean` EJB. The `webservices.xml` file in [Example 10-3](#) also tells JBoss where to find the WSDL and JAX-RPC Mapping files in the EJB JAR file.

Example 10-3. webservices.xml

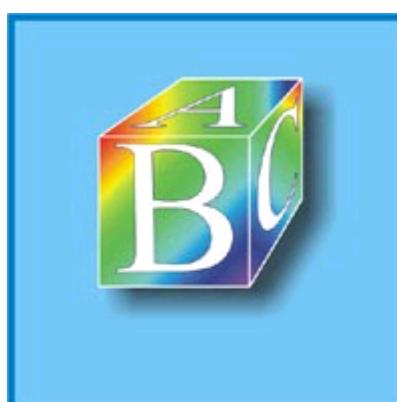
```
<?xml version="1.0" encoding="UTF-8"?>

<webservices
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://www.ibm.com/webservices/xsd/j2ee_web_services_1_1.xsd"
  version="1.1">
  <webservice-description>

    <webservice-description-name>InventoryService</webservice-description-name>
    <wsdl-file>META-INF/wsdl/InventoryService.wsdl</wsdl-file>

    <jaxrpc-mapping-file>META-INF/inventory-mapping.xml</jaxrpc-mapping-file>
      <port-component>
        <port-component-name>Inventory</port-component-name>
        <wsdl-port>InventoryEndpointPort</wsdl-port>
        <service-endpoint-interface>
          com.jbossatwork.ws.InventoryEndpoint
        </service-endpoint-interface>
        <service-impl-bean>
          <ejb-link>InventoryFacade</ejb-link>
        </service-impl-bean>
      </port-component>
    </webservice-description>
  </webservices>
```

We've registered the Web Service endpoint and told the server about the JAX-RPC mapping and WSDL files, and now we need to create these extra descriptors.

[◀ PREV](#)[NEXT ▶](#)

ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

10.8. JAX-RPC Mapping File

The J2EE-standard JAX-RPC mapping file helps the JAX-RPC compiler map Java objects to WSDL objects. If the Java objects are complex, the JAX-RPC and WSDL files also will be complex. [Example 10-4](#) is the `inventory-mapping.xml` JAX-RPC mapping file that the JAW Motors application uses.

Example 10-4. `inventory-mapping.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<java-wsdl-mapping xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://www.ibm.com/webservices/xsd/j2ee_jaxrpc_mapping_1_1.xsd"
    version="1.1">

    <package-mapping>
        <package-type>com.jbossatwork.ws</package-type>
        <namespaceURI>http://localhost:8080/jbossatwork-ws/types</namespaceURI>
    </package-mapping>
    <package-mapping>
        <package-type>com.jbossatwork.ws</package-type>
        <namespaceURI>http://localhost:8080/jbossatwork-ws</namespaceURI>
    </package-mapping>
    <java-xml-type-mapping>
        <java-type>com.jbossatwork.dto.CarDTOArray</java-type>
        <root-type-qname
            xmlns:typeNS="http://localhost:8080/jbossatwork-ws/types">
            typeNS:CarDTOArray
        </root-type-qname>
        <qname-scope>complexType</qname-scope>
        <variable-mapping>
            <java-variable-name>cars</java-variable-name>
            <xml-element-name>cars</xml-element-name>
        </variable-mapping>
    </java-xml-type-mapping>
    <java-xml-type-mapping>
        <java-type>com.jbossatwork.dto.CarDTO</java-type>
        <root-type-qname
            xmlns:typeNS="http://localhost:8080/jbossatwork-ws/types">
            typeNS:CarDTO
        </root-type-qname>
        <qname-scope>complexType</qname-scope>
        <variable-mapping>
            <java-variable-name>id</java-variable-name>
            <xml-element-name>id</xml-element-name>
        </variable-mapping>
        <variable-mapping>
            <java-variable-name>make</java-variable-name>
            <xml-element-name>make</xml-element-name>
        </variable-mapping>
        <variable-mapping>
            <java-variable-name>model</java-variable-name>
            <xml-element-name>model</xml-element-name>
        </variable-mapping>
        <variable-mapping>
            <java-variable-name>modelYear</java-variable-name>
            <xml-element-name>modelYear</xml-element-name>
        </variable-mapping>
        <variable-mapping>
            <java-variable-name>status</java-variable-name>
            <xml-element-name>status</xml-element-name>
        </variable-mapping>
    </java-xml-type-mapping>
</java-wsdl-mapping>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

10.9. WSDL File

The WSDL file describes a Web Service interface along with its parameters and registers the web service with JBoss. If you thought the previous descriptors were tedious and painful to look at, then you're in for a treatthe WSDL file is much worse. [Example 10-5](#) is the `InventoryService.wsdl` WSDL file used to deploy the Inventory Web Services.

Example 10-5. `InventoryService.wsdl`

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="InventoryService"
targetNamespace="http://localhost:8080/jbossatwork-ws"
xmlns:tns="http://localhost:8080/jbossatwork-ws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://localhost:8080/jbossatwork-ws/types"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <schema targetNamespace="http://localhost:8080/jbossatwork-ws/types"
      xmlns:tns="http://localhost:8080/jbossatwork-ws/types"
      xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="CarDTOArray">
        <sequence>
          <element name="cars" type="tns:CarDTO" nillable="true" minOccurs="0"
            maxOccurs="unbounded"/></sequence></complexType>
      <complexType name="CarDTO">
        <sequence>
          <element name="id" type="int"/>
          <element name="make" type="string" nillable="true"/>
          <element name="model" type="string" nillable="true"/>
          <element name="modelYear" type="string" nillable="true"/>
          <element name="status" type="string"
            nillable="true"/>
        </sequence>
      </complexType>
    </schema>
  </types>
  <message name="InventoryEndpoint_findAvailableCars"/>
  <message name="InventoryEndpoint_findAvailableCarsResponse">
    <part name="result" type="ns2:CarDTOArray"/></message>
  <portType name="InventoryEndpoint">
    <operation name="findAvailableCars">
      <input message="tns:InventoryEndpoint_findAvailableCars"/>
      <output message="tns:InventoryEndpoint_findAvailableCarsResponse"/>
    </operation>
  </portType>
  <binding name="InventoryEndpointBinding" type="tns:InventoryEndpoint">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="rpc"/>
    <operation name="findAvailableCars">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"
namespace="http://localhost:8080/jbossatwork-ws"/>
      </input>
      <output>
        <soap:body use="literal"
namespace="http://localhost:8080/jbossatwork-ws"/>
      </output>
    </operation>
  </binding>
</definitions>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

10.10. Set the Web Service URL

We want to use a meaningful URL that matches our Web Services namespace `jbossatwork-ws`. So we modify `jboss.xml` in [Example 10-6](#) (the JBoss-specific EJB deployment descriptor) as follows.

Example 10-6. jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"
          "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">

<jboss>

  <enterprise-beans>
    ...
    <session>
      <ejb-name>InventoryFacade</ejb-name>
      <port-component>
        <port-component-name>Inventory</port-component-name>
      <port-component-uri>jbossatwork-ws/InventoryService</port-component-uri>
      </port-component>
    ...
    </session>
    ...
  </enterprise-beans>
  ...
</jboss>
```

The `<port-component>` element and its sub-elements tell JBoss to deploy our WSDL to the `jbossatwork-ws` namespace at the following URL:

<http://localhost:8080/jbossatwork-ws/InventoryService?wsdl>

Although modifying the `jboss.xml` file to set the URL is helpful, it is purely optional. You could successfully deploy a Web Service without changing `jboss.xml`.

At this point we've shown all the Web Services -related deployment descriptors. We now have to upgrade `InventoryFacadeBean` to expose its `findAvailableCars()` method as a Web Service.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

10.11. Modifying the InventoryFacadeBean EJB

[Example 10-7](#) shows the new and upgraded XDoclet tags in the `InventoryFacadeBean` EJB that support Web Services deployment.

Example 10-7. `InventoryFacadeBean.java`

```
/*
 * @ejb.bean
 *   name="InventoryFacade"
 *   ...
 *   view-type="all"
 *   ...
 *
 * @wsee.port-component
 *   name="Inventory"
 *   wsdl-port="InventoryEndpointPort"
 *   service-endpoint-interface="com.jbossatwork.ws.InventoryEndpoint"
 *   service-endpoint-bean="com.jbossatwork.ejb.InventoryFacadeBean"
 *
 * @ejb.interface
 *   service-endpoint-class="com.jbossatwork.ws.InventoryEndpoint"
 *
 */
public class InventoryFacadeBean implements SessionBean {
    ...

    /**
     * @ejb.interface-method
     *   view-type="all"
     *   ...
     */
    public CarDTOArray findAvailableCars( ) throws EJBException {
        CarDTOArray carDTOArray = new CarDTOArray( );
        CarDTO[ ] cars = (CarDTO[ ]) listAvailableCars( ).toArray(new
CarDTO[0]);

        carDTOArray.setCars(cars);

        return carDTOArray;
    }

    /**
     * @ejb.interface-method
     *   view-type="both"
     *   ...
     */
    public List listAvailableCars( ) throws EJBException {
        ...
    }
}
```

We've added a new method called `findAvailableCars()` that we're exposing as a Web Service. This method doesn't do muchit wraps the call to the `listAvailableCars()` method (which won't be used as a Web Service) and converts its return value (a `java.util.List`) to a `CarDTOArray`. We added the `findAvailableCars()` method because Java 2 Collections are incompatible with J2EE 1.4 Web Services, and we didn't want to change the existing `listAvailableCars()` method. See the "[Web Services Data Types](#)" and "[Web Services and Collections](#)" sections for more details.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

10.12. Web Services Deployment

The new files go into the EJB JAR file as follows:

- In addition to the `ejb-jar.xml`, `jboss.xml`, and JAR Manifest files, the `META-INF` directory now contains `webservices.xml` and `inventory-mapping.xml` (the JAX-RPC mapping file). The WSDL file, `InventoryService.wsdl`, is in `META-INF/wsdl`.
- The `com/jbossatwork/ws` directory holds the `InventoryService` (Service Endpoint Interface) class file.
- Everything else remains unchanged.

The new `CardDTOArray` object class has been added to the `com.jbossatwork.dto` package, so its class file resides in the Common JAR's `com/jbossatwork/dto` directory.

At this point, we've shown all the Web Services-related deployment descriptors and upgraded `InventoryFacadeBean`. Let's show how to generate the descriptors and package the Web Service for deployment with Ant, XDoclet, and JWSDP.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

10.13. Automating Web Services Deployment

As we've seen so far, deploying a Web Service requires writing very little code (just the Service Endpoint Interface), but it adds a complex set of deployment issues:

- Create three new descriptors (`webservices.xml`, the JAX-RPC Mapping file, and the WSDL file)
- Modify both EJB deployment descriptors (`ejb-jar.xml` and `jboss.xml`).

Although it's possible to develop everything by hand, it's not very practical because the new deployment descriptors are interrelated, and the XML elements are tedious and error-prone. In previous chapters, we've automated everything with Ant and XDoclet, but the current XDoclet version, XDoclet 1.2.3, falls short when it comes to Web Services. XDoclet has Ant tasks that are supposed to generate the JAX-RPC Mapping and WSDL files, but these XDoclet tasks don't work properly. Bug reports have been submitted to the XDoclet project, and we hope to see a resolution to these issues in the next production version of XDoclet. As you'll see in the "Web Services Ant Script" section, we'll add another technology to our toolkitthe Java Web Services Developer's Pack (JWSDP)to complete our deployment by generating the JAX-RPC Mapping and WSDL files. Even though XDoclet doesn't do everything, we still use it to generate the Service Endpoint Interface, modify `ejb-jar.xml`, and generate `webservices.xml`.

Before you can use the JWSDP from our Ant build script, you need to download JWSDP 1.5 from <http://java.sun.com/webservices/downloads/webservicespack.html> and add the JAR files to your `CLASSPATH` by doing one of the following:

- In the Ant build script below, set the `jwsdp.lib.dir` property to your JWSDP 1.5 installation.
- Copy the `lib`sub-directory from your Axis 1.1 installation to `/Library/jwsdp-1.5/` (the `jwsdp.lib.dir` property in the Ant build script currently points to `/Library/jwsdp-1.5`).

10.13.1. Web Services Ant Script

We now modify the EJB deployment process to include Web Service deployment. [Example 10-12](#) shows the upgraded `build.xml` script from the `ejb` sub-project.

Example 10-12. ejb/build.xml

```
...
<property name="jwsdp.lib.dir" value="/Library/jwsdp-1.5"/>
...
<target name="run-ejbdoclet" ...>
  <ejbdoclet ...>
    ...
    <service-endpoint/>
    ...
  </ejbdoclet>
  <!-- Fix problems with XDoclet-generated Service Endpoint Interface -->
  <replace
    file="${gen.source.dir}/com/jbossatwork/ws/InventoryEndpoint.java">
    <replacetoken><! [CDATA[throws javax.ejb.EJBException,
java.rmi.RemoteException]]></replacetoken>
    <replacevalue><! [CDATA[throws
java.rmi.RemoteException]]></replacevalue>
  </replace>
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

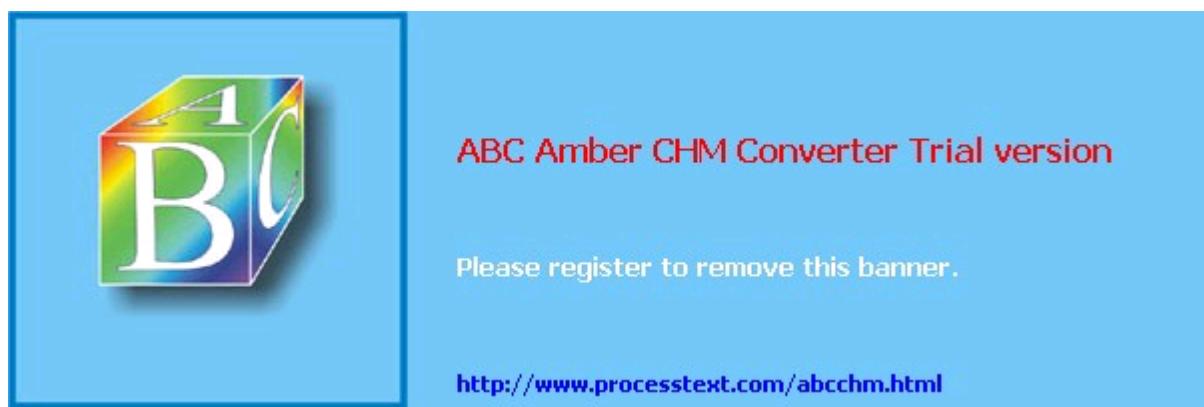
10.14. J2EE Web Services Checklist

Before we move on to test the JAW Motors application's new Web Service functionality, let's recap what we've done to implement a J2EE 1.4 Web Service:

- Created the `InventoryEndpoint` Service Endpoint Interface (SEI) to expose EJB methods as Web Service operations
- Added a `<service-endpoint>` element to `ejb-jar.xml` to tell JBoss that the `InventoryFacadeBean` EJB implements the `InventoryEndpoint` Service Endpoint Interface
- Generated the `webservices.xml` to register the Web Service and tie the `InventoryEndpoint` Service Endpoint Interface to an implementation the `InventoryFacadeBean`
- Created the JAX-RPC Mapping File to define JAX-RPC type mappings for the parameters and return values for the `InventoryEndpoint`'s method
- Generated the `inventory-mapping.xml` WSDL file to define the Web Service and tie it to XML Schema data types
- Set the Web Service URL by modifying `jboss.xml` with the Inventory Web Service URL
- Modified the `InventoryFacadeBean`:
 - Added Web Services-related XDoclet tags
 - Added the `findAvailableCars()` method and used an XML Schema-compatible data type%
- Upgraded deployment by modifying the Ant build script:
 - Fixed XDoclet-generated descriptors
 - Modified the Web Service URL in `jboss.xml`
 - Used JWSDP to generate the JAX-RPC and WSDL files

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

10.15. Testing Web Services Deployment

It's taken us a while to get here, but now that we have the core infrastructure in place to deploy a Web Service, let's test our deployment. Here are the steps to build and deploy the application:

- Type `ant` in the root directory of `ch10` to build the project.
- Shut down JBoss so the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBOSS_HOME/server/default/deploy` directory.
- Start JBoss back up.

You should see the following output in the JBoss console:

```
...
23:08:11,921 INFO  [WSDLFilePublisher] WSDL published to: file:/C:/jboss-
4.0.2/server/default/data/wsdl/jaw.ear/ejb.jar/InventoryService.wsdl

23:08:12,182 INFO  [AxisService] WSDD published to: C:\jboss-
4.0.2\server\default\data\wsdl\jaw.ear\ejb.jar\Inventory.wsdd

23:08:12,632 INFO  [AxisService] Web Service deployed:
http://localhost:8080/jbossatwork-ws/InventoryService
...
```

Now point your browser to <http://localhost:8080/ws4ee> to see the JBossWS page as shown in [Figure 10-2](#).

Figure 10-2. JBossWS page Welcome to JBossWS

This is the JBoss J2EE-1.4 compatible webservice implementation based on Axis.

- Validate the local installation's configuration
- View the list of deployed Web services

Clicking on the "View the list of deployed Web services" link takes you to the JBoss Deployed Web Services Page, as depicted in [Figure 10-3](#).

Figure 10-3. JBoss Deployed Web Services page And now... Some Services

- Version (wsdl)
 - getVersion
- ch10.ear/ejb.jar#Inventory (wsdl)
 - findAvailableCars

At this point, you'll see our Web Service (`findAvailableCars`) listed under `jaw.ear/ejb.jar#Inventory`. Click on the `wsdl` link, and you'll see the WSDL for our Web Service.

Now that we've successfully deployed our Web Service and viewed the WSDL, keep JBoss running. We now move on to develop an external client that calls the `findAvailableCars` Web Service.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

10.16. Web Services Client

We viewed the Web Service's WSDL, so we know that it deployed properly. However, now we need to make sure that the Web Service really works. So, we're going to add an external client application that tests the Web Service. Running a client against our Web Service will flush out any Web Service-related issues like custom data type serialization/de-serialization errors (caused by using data types that aren't compatible with XML Schema data types).

For our client, we'll use a well-known API for invoking the Inventory Web Service methods. Here are some of our choices:

JNDI

We could add `<service-ref>` elements to our EJB and client deployment descriptors so an external client could use a JNDI name like `java:comp/env/service/Inventory` to look up our service endpoint and invoke its methods.

JAX-RPC

We could use JAX-RPC calls that look up the service by using its port name from the WSDL, and then specify serializers/deserializers to handle the custom data types.

Apache Axis

We could download the WSDL from <http://localhost:8080/jbossatwork-ws/InventoryService?wsdl> and use the Apache Axis toolkit to generate proxy and custom data type objects that communicate with the Web Service.

Perl, Python, C#

Clients could have been written in any language that supports Web Services, but since this is a Java book, we decided to stick with Java.

JNDI and JAX-RPC are too Java/J2EE-centric, so these approaches don't provide a good test for the interoperability of our Web Service. Besides, the main reason for using Web Services is to enable clients to seamlessly use a service, regardless of the programming language used on either side. We chose Axis because:

- Axis code generation is based on WSDL, making it platform-neutral.
- The generated proxy and custom data type objects make it much easier to write the client.

Our client is written in Java, but non-Java clients written in other languages like C# could also take our WSDL, generate a proxy using their own language-specific tools, and use our Web Service. WSDL completely decouples a Web Service from its clients because:

- Each client generates a proxy and any associated custom data types from the WSDL into its native language.
- There are no language- or platform-specific dependencies between a Web Service and its clients.

Before we develop a Web Service client, let's explore the new directory structure we'll use for our Web Service client development environment.

10.16.1. Exploring the New Directory Structure

In previous chapters, we had subdirectories for building the Common JAR (`common`), the database (`sql`), the web application (`webapp`), and EJBs (`ejb`).

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

10.17. Implementing a Web Service Client

To implement a Web Service Client, we need to do the following:

- Download the WSDL from <http://localhost:8080/jbossatwork-ws/InventoryService?wsdl>
-
- Generate Web Service proxy and custom data type objects by using the Axis WSDL2Java Ant task.
- Write a Web Service client that uses the Axis-generated proxy and custom data type objects to call the Web Service.

After downloading the WSDL and saving it to `client/InventoryService.wsdl` under the `ch10` project root directory, we use the Axis 1.1 WSDL2Java Ant task in the client `build.xml` file to generate our Web Service proxy objects and custom data types ([Example 10-14](#)).

Example 10-14. client/build.xml

```

...
<property name="axis.lib.dir" value="/Library/axis-1_1/lib"/>
...

<path id="axis.classpath">
    <fileset dir="${axis.lib.dir}">
        <include name="**/*.jar"/>
    </fileset>
</path>
...

<target name="run-wsdl2java" description="Generates WS proxy code from
WSDL">
    <path id="wsdl2java.task.classpath">
        <path refid="axis.classpath"/>
    </path>

    <taskdef name="wsdl2java"
            classname="org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask">
        <classpath refid="wsdl2java.task.classpath"/>
    </taskdef>

    <mkdir dir="${gen.source.dir}" />

    <wsdl2java output="${gen.source.dir}"
               url="InventoryService.wsdl"
               verbose="true">

        <mapping namespace="http://localhost:8080/jbossatwork-ws"
                 package="com.jbossatwork.client"/>

        <mapping namespace="http://localhost:8080/jbossatwork-ws/types"
                 package="com.jbossatwork.client"/>

    </wsdl2java>
</target>
```

The `<run-wsdl2java>` target uses the Axis `<wsdl2java>` task to generate the Web Service proxy objects (`InventoryServiceLocator`, `InventoryService`, and `InventoryEndpoint`) and custom data types (`CarDTOArray` and `CarDTO`) for the client. The `<mapping>` elements map the namespace from the WSDL to our Java package name `com.jbossatwork.client`. The namespace for each `<mapping>` element comes from the WSDL:

- <http://localhost:8080/jbossatwork-ws> is the WSDL namespace for the proxy objects.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

10.18. Web Service Client Checklist

Before we move on to test the Web Service Client, let's recap what we've done:

- Downloaded the Web Service's WSDL from
<http://localhost:8080/jbossatwork-ws/InventoryService?wsdl>
- Generated Web Service Web Service proxy and custom data type objects by using the Axis WSDL2Java Ant task
- Wrote a Web Service client that uses the Axis-generated proxy and custom data type objects to call the Web Service.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

10.19. Testing the Web Service Client

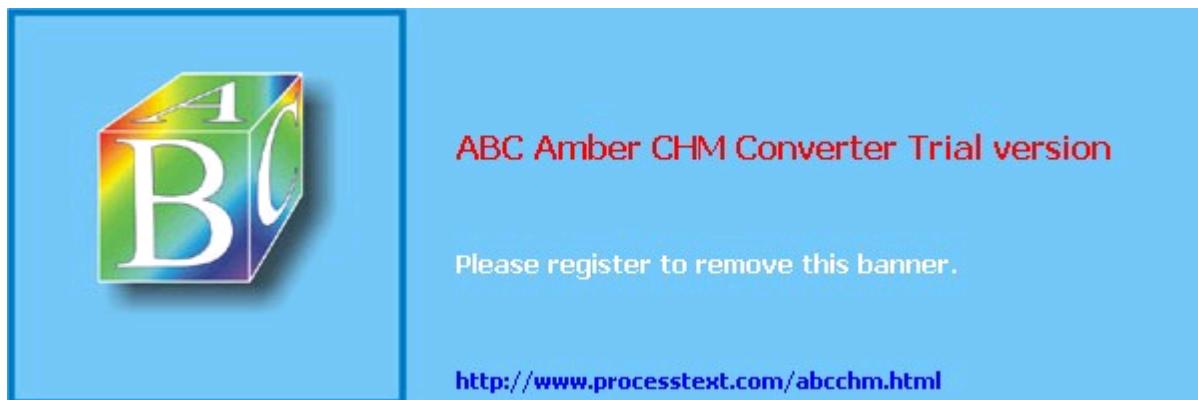
We've generated Web Service proxy code and written a client to call the `findAvailableCars` Web Service. Go to the `ch10/client` sub-directory, and compile and run the client by typing: `ant run-client`.

Ant runs the client (by `MyAxisClient`'s `main()` method) after generating and compiling the client code. You should see the following output in the client console:

```
...
[java] Finding InventoryService ...
[java] Getting InventoryEndpoint ...
[java] Getting Cars ...
[java] Year = [2005], Make = [Toyota], Model = [Camry], status =
[Available]
[java] Year = [1999], Make = [Toyota], Model = [Corolla], status =
[Available]
[java] Year = [2005], Make = [Ford], Model = [Explorer], status =
[Available]
```

[◀ PREV](#)

[NEXT ▶](#)



10.20. Final Thoughts on J2EE 1.4 Web Services

Here are the lessons we learned from deploying a J2EE 1.4 Web Service:

- Adding Web Services to your application doesn't have a huge impact on your server-side code, but it increases the complexity of your deployment.
- If you already have a Stateless Session Bean, you only have to modify and add some XDoclet tags to expose its methods as Web Services.
- Make sure that the Java classes you're using as parameters and return values follow the Java Beans conventions.
- Java 2 Collections and arrays of custom data types are incompatible with Web Services, so wrap an array of custom data types in a Java Bean.
- If your Web Service uses custom data types as parameters or return values, make sure you use JBoss 4.0.1sp1 or higher.
- Set your Web Service URL to something meaningful in `jboss.xml`.
- The tools that generate Web Services server-side deployment artifacts are still maturing.
- XDoclet 1.2.3 has limited support for Web Services. Use XDoclet to:
 - Generate the `<service-endpoint>` element in `ejb-jar.xml`.
 - Create `webservice.xml`.
 - Generate the Service Endpoint Interface.%
- Due to XDoclet's limitations, use JWSDP's `wscompile` tool to generate the JAX-RPC Mapping and WSDL files.
- Check your Web Service deployment by viewing the JBossWS Page (<http://localhost:8080/ws4ee>) on your JBoss instance.
- On the client side, use the WSDL to generate proxy and custom data type objects in your native programming language to communicate with a Web Service. Using WSDL-based client code tests the interoperability of your Web Service.
- Even if you're using Java, using the Axis toolkit to invoke a Web Service gives you a simple, elegant Object Oriented interface that hides the low-level API calls.
- If you're using Java 5, make sure you use Axis 1.2 or higher.
- If you're using J2SE 1.4, use Axis 1.1.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

10.21. Conclusion

In this chapter we showed how to expose a portion of the JAW Motors application as a Web Service so JAW Motors can share its inventory with its online trading partners and work with non-Java clients. We exposed an EJB as a Web Service by using Ant, XDoclet, and JWSDP to deploy it on JBoss. We finished by writing an Axis client that invoked our Web Service.

◀ PREV

NEXT ▶



◀ PREV

NEXT ▶

10.22. Congratulations!

Congratulations you did it! You started off this book by deploying an EAR and serving up simple content with hardcoded data for demonstration purposes. Then you progressed to using a JDBC connection to get data from JBoss' built-in HyperSonic database. You wired up the database using Hibernate to make the database access easier, more platform-independent, and to more easily automate the application. You added Session Beans to the application to manage transactions. Then you used JMS to send a message to a Credit Card approval system and sent an email with JavaMail to indicate the success or failure of the credit check. You then added security to the application to prevent unauthorized access to certain portions of the application. You finished by exposing a portion of the application as a Web Service so JAW Motors could share its inventory with its online trading partners. We hope that you can take what you've learned in this book and apply it to your real-world JBoss projects.

◀ PREV

NEXT ▶



Appendix A. ClassLoaders and JBoss

When a Java application references Java classes, the Java Virtual Machine (JVM) uses a ClassLoader to load them into memory. The Delegation Model was introduced in Java 2 and organizes ClassLoaders into the following tree/hierarchy by using parent/child relationships, as shown in [Figure A-1](#).

Figure A-1. Standard J2SE ClassLoader hierarchy



The J2SE ClassLoaders do the following:

- The Bootstrap (also called the *primordial*) ClassLoader has no parent, is the root of the ClassLoader tree, and loads core Java classes (`java.*`) into the JVM.
- The Extension ClassLoader loads extension classes:
 - Classes that extend core Java classes `javax.*`.
 - Classes from the Java Runtime Environment (JRE) `lib/ext` directory in the standard J2SE installation.
- The System (also called the Application) ClassLoader loads classes and JARs from the system `CLASSPATH`, the `CLASSPATH` environment variable and the `classpath` argument on the `java` command line.

If the current ClassLoader previously loaded a class, then the ClassLoader returns the class to the client. If a class has not been previously loaded, then according to the Java specification, a ClassLoader must defer (or delegate) to its parent before trying to load the class itself. For example, if an application references `java.lang.String`, the System ClassLoader delegates to the Extension ClassLoader, which in turn defers to the Bootstrap ClassLoader to load the String class. The child ClassLoader gets a chance to load a class only if the parent hasn't already loaded the class or couldn't load the class. A class is loaded only once per ClassLoader.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

A.1. Namespaces

The JVM recognizes a class by its runtime identity—the class' full name (the package plus class name), along with the instance of the ClassLoader that instantiated the class. Each ClassLoader has its own namespace consisting of the classes it loads, and each fully qualified class name MUST be unique within that namespace. This naming convention is required by the J2EE specification. Two ClassLoaders could each load the same class, and the JVM would treat each class as a distinct type. Thus the JVM considers class `com.jbossatwork.util.TextEmail` in ClassLoader 1 different from the `com.jbossatwork.util.TextEmail` in ClassLoader 2 because they have different runtime identities based on the ClassLoader name. We'll see why this distinction is important when we get to the section on the JBoss ClassLoaders.

◀ PREV

NEXT ▶



 PREV

NEXT 

A.2. Class Loading in the J2EE

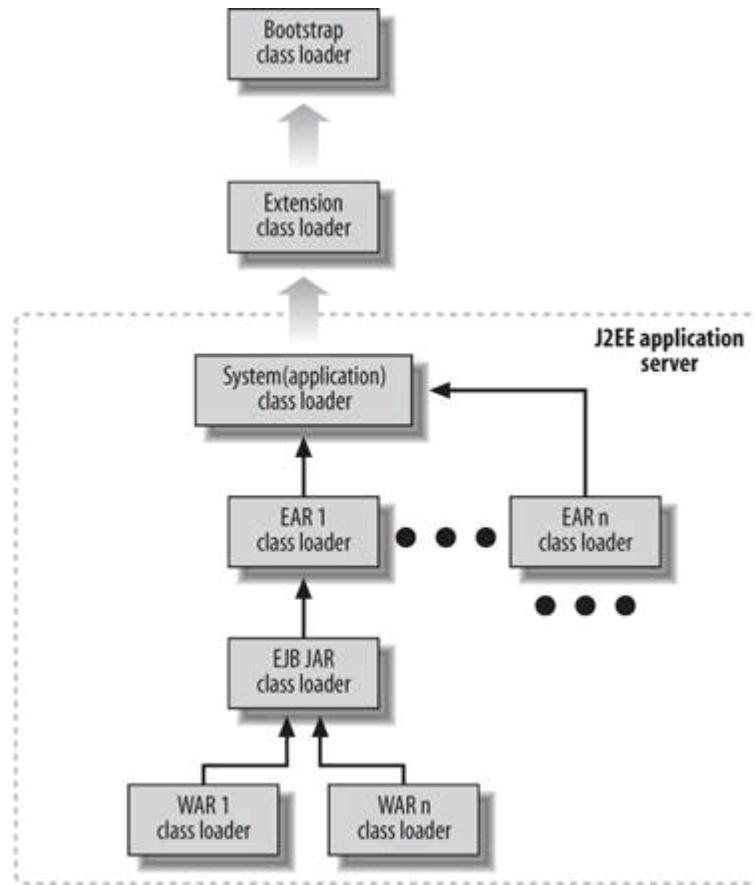
Class Loading is one of the least understood aspects of J2EE deployment. Although not officially part of the J2EE specification, *most* application servers use a strategy similar to the ClassLoader hierarchy in [Figure A-2](#) to support J2EE component deployment.

The J2EE-based ClassLoaders work as follows:

- The EAR ClassLoader loads the classes from JARs contained in the EAR file.
- The EJB ClassLoader loads EJBs and related classes that reside in the EJB JAR file.
- The WAR ClassLoader loads the web-specific classes and JARs in the WAR file's `WEB-INF/{class, lib}` directories.

Because each ClassLoader has no access to the classes loaded by its children or siblings, the classes in the WAR file are not visible to EJBs. Of course EJBs shouldn't try to reference things in the WAR in the first place because it is poor architecture. Each layer should not know about the invoking layer, and you can't assume that a web-based presentation layer will always be in your application.

Figure A-2. Generic J2EE Application Server Class Loader Hierarchy



The J2EE specification is vague concerning the deployment order of the J2EE modules. Many application servers deploy the JARs and classes in the EAR, and then the classes in the EJB JAR, and finally, the classes and JARs in the WAR. In the JAW Motors application, the Common JAR strategy works because the EAR contains the Common JAR, so the application-specific dependency classes and the third party utility JARs get deployed first, then the EJBs in the EJB JAR are loaded, and the web components from the WAR which depend on the EJBs and common classes and JARs deploy last.

Here is a practical example that uses the class-loading scenario from [Figure A-3](#). In the JAW Motors application, a JSP invokes a Servlet, which in turn uses an EJB; both the Servlet and the EJB use Log4J. Since the JSP and Servlet are packaged in the WAR, the WAR ClassLoader finds and loads the JSP and Servlet. When the Servlet instantiates the EJB, the WAR ClassLoader cannot find the EJB, so WAR ClassLoader defers to its parent the EJB ClassLoader. The EJB JAR contains the EJB, so the EJB ClassLoader finds and loads the EJB on behalf of the

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

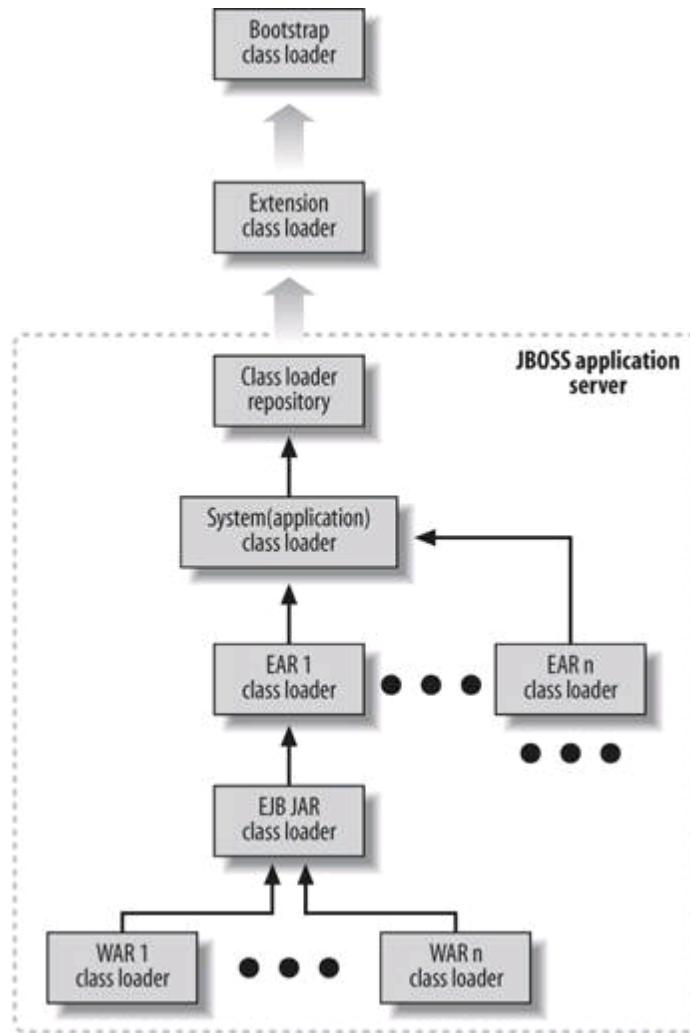
 PREV

NEXT 

A.3. Class Loading with JBoss

JBoss' class loading scheme is similar to the strategy used by other J2EE application servers, as shown in [Figure A-3](#).

Figure A-3. JBoss ClassLoader Hierarchy



JBoss' ClassLoader hierarchy differs from the strategy used by other J2EE application servers in the following ways:

The ClassLoader Repository

The ClassLoader Repository contains all classes loaded by a JBoss instance, including:

- JBoss' internal boot classes, including its J2EE component implementations
- Any classes or JARs specified on the command line when starting JBoss
- All classes or JARs from each deployed application

By default, only one ClassLoader Repository covers the entire server. But you are free to declare any number of repositories and associate any deployed applications with a repository. To keep deployment simple, the JAW Motors application uses the default ClassLoader Repository.

Cross-referencing between EJB JAR and WAR files

Within an EAR, an EJB could access classes and/or properties packaged in a WAR file (Log4J for example). The converse is true as wellweb components could access resources bundled in an EJB JAR. However, we do not recommend this practice

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

A.4. Common ClassLoader Issues

ClassLoader issues are difficult to find and take a long time to debug. They fall into one of two categories:

Not enough visibility

You'll see one of the following exceptions:

- `ClassNotFoundException`

Java API methods such as `Class.forName()` or `ClassLoader.loadClass()` throw a `ClassNotFoundException`. This exception happens when a class loader tries to load a class and can't find the class. Here are some possible causes:

- A JAR or directory for the class is not available, so the ClassLoader asked to load the class, or to its parent(s).
- The wrong ClassLoader is used to load the class.

NoClassDefFoundError

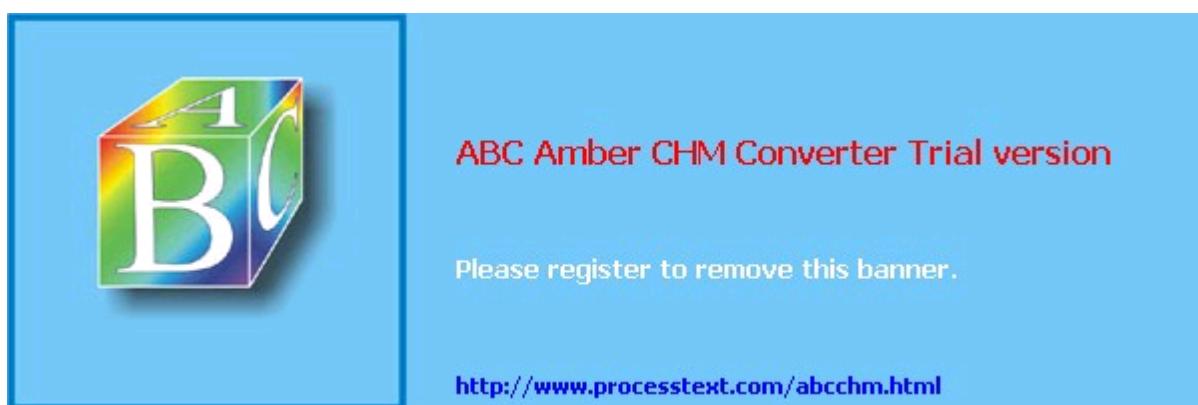
A `NoClassDefFoundError` has the same causes as the `ClassNotFoundException`, but there is an additional reason a class depends on other classes that are inaccessible from the current ClassLoader. The other classes may reside only in a child or sibling ClassLoader, neither of which is available to the current ClassLoader.

Too much visibility

This happens when you have a duplicate class and the problem manifests as a `ClassCastException`. For example, you could include the same JAR file (bundled with several libraries) more than once in your deployment.

◀ PREV

NEXT ▶



A.5. ClassLoader Options

In the Java API, you can explicitly use either the System (or Application), Current, or the Thread Context ClassLoader.

You can access the System ClassLoader by calling `ClassLoader.getSystemClassLoader()`. In most J2EE application servers, the System ClassLoader is too high in the hierarchy and will not find the resources packaged in your application's EJB JAR or WAR. In JBoss, using the System ClassLoader is still a problem because the ClassLoader Repository holds references to all deployed applications, so you could easily have a naming conflict if more than one application uses the same class.

The Current ClassLoader is the ClassLoader that loaded the class that contains the method that's currently executing. `Class.getResource()` and the one-parameter version of `Class.forName()` use the Current ClassLoader by default. The Current ClassLoader is a better choice than the System ClassLoader, but there are still problems with using the Current ClassLoader:

- You may not know who calls your class, and the current ClassLoader could be up too high in the ClassLoader to find the resources that your class needs.
- In JBoss, you could have the same naming conflict with Class Loader Repository that you had with the System ClassLoader.

You gain access to the current Thread Context ClassLoader by calling `Thread.currentThread().getContextClassLoader()`. The Thread Context ClassLoader is the ClassLoader used by the creator of the Thread that runs your code. The Thread Context ClassLoader works in a way that's contrary to the Delegation Model by enabling a parent ClassLoader to access classes from any of its child ClassLoaders. Sometimes a parent ClassLoader needs to see classes that one of its child ClassLoaders instantiates at runtime. Use the Thread Context Class Loader for the following reasons:

- In JBoss, you're guaranteed to load the class or property file from your application by using the Thread Context ClassLoader. Even though the JBoss ClassLoader Repository may have the same class or property from several applications, the Thread Context ClassLoader picks the class or property that belongs to your application.
- The EJB specification forbids EJBs to use the Current Class Loader, and since the System Class Loader isn't a workable option, you're left with the Thread Context ClassLoader. See the Programming Restrictions section in the EJB specification for further details.

Here are a couple of practical uses of the Thread Context ClassLoader:

- If you have a factory method that uses `Class.forName()` to instantiate classes dynamically, pass the Thread Context ClassLoader as a parameter to `Class.forName()`.
- To load a Properties file in your application's `CLASSPATH`, call `Thread.currentThread().getContextClassLoader().getResourceAsStream()` to find the Properties file, and use the resulting `InputStream` when calling `Properties.load()`.



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)

A.6. Solving ClassLoader Issues

The best way to solve ClassLoader issues is to reduce their likelihood by following good deployment practices:

- Know the basics of ClassLoaders and how your particular deployment environment uses them. Even though it sounds boring, make a deployment plan and stick with it. Know the dependencies for each J2EE module. Then factor out the common classes and utility JARs into a Common JAR that resides in the EAR. When you add new dependency classes or third party utility JARs, update the plan and refactor. Planning where your classes and utilities reside reduces the chance that you'll have `ClassNotFoundException` and `NoClassDefFoundError` issues.
- Make sure that there are no duplicate classes or JAR files. Third party libraries typically cause this problem because they each may have their own copy of a particular JAR file. For example, many Apache libraries have their own copy of files such as Log4J and Commons Lang. Make sure there is only one copy of these classes or JAR files across the entire EAR deployment. This process is tedious and requires upgrading Ant build scripts, but it saves serious headaches later on. Without duplicate classes or JARs, you're less likely to see problems like the `ClassCastException`.
- Encapsulate/Minimize Visibility to avoid `ClassCastException`s.
 - Put web-specific classes in the WAR file's *WEB-INF/classes* directory and web-specific third party JARs (like Struts, for example) in the WAR file's *WEB-INF/lib* directory. There's no need to put web-specific classes and JARs in the Common JAR because other J2EE modules don't depend on them.
 - Put extra EJB-specific utility classes and dependent JAR files in EJB JAR's root directory because the web components don't need to access them.
 - Use a Common JAR that contains all utility classes and JARs used in both the EJB JAR and the WAR.
- Use the correct ClassLoader to avoid the `ClassNotFoundException` and `NoClassDefFoundError`. Use the Thread Context ClassLoader (`Thread.currentThread().getContextClassLoader()`). If you use `Class.forName()`, pass in the Thread Context ClassLoader as a parameter. To load a Properties file in your application's `CLASSPATH`, use the Thread Context ClassLoader to get an `InputStream` for loading the Properties file.

[◀ PREV](#)[NEXT ▶](#)The logo features a 3D cube with a rainbow gradient. The letters 'A' and 'B' are embossed on the front face of the cube.

ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

A.7. Conclusion

In this Appendix, we covered J2SE and J2EE ClassLoaders. We then showed how Class Loading works in JBoss, and finished by discussing how to reduce the probability of ClassLoader problems in your applications.

◀ PREV

NEXT ▶

ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

Appendix B. Logging and JBoss

Most J2EE applications use some form of logging or tracing API that stores messages to a persistent medium (such as DBMS or file). Logging provides the following benefits:

Debugging

Developers generate log messages to debug their programs.

Reviewing

System personnel examine log messages to check for problems.

Auditing

Security personnel can review log messages to see what actions a user performed in the system.

This chapter covers two of the most common logging APIs and how to use them with JBoss:

- Jakarta Commons Logging (JCL) API
- Apache Log4J

◀ PREV

NEXT ▶



 PREV

NEXT 

B.1. Jakarta Commons Logging (JCL) API

The Jakarta Commons Logging (JCL) package from the Apache Jakarta project provides a standard interface to the various logging libraries, hiding the details of the logging implementation from an application. Many logging APIs are available, and at some point an application may need to change to another logging technology. The main benefit of the JCL is that it enables developers to switch between logging implementations without impacting their code.

B.1.1. Using the Apache JCL

[Example B-1](#) is an example from the JAW Motors `InitServlet` that uses the JCL to print out a log message.

Example B-1. `InitServlet.java`

```
...
import org.apache.commons.logging.*;
...
public class InitServlet extends GenericServlet {
    private Log log = LogFactory.getLog(InitServlet.class);
    private ServletContext servletContext;
    public void init( ) {
        ...
        log.info("Testing Logging Setup ...");
    }
}
```

The call to `LogFactory.getLog()` creates a concrete implementation of the `org.apache.commons.logging.Log` interface so we can log messages. The `init()` method then uses the `Log` instance to log a debug message by calling `log.debug()` this method then does the real work of logging the message to its destination. See the next section for more information about `org.apache.commons.logging.Log` interface. See the [Initialization Servlet](#) section for more information on the `InitServlet`.

B.1.2. Using a Logging Implementation

A logging package must implement the `Log` interface to work with the JCL, which comes bundled with four usable concrete logger instances that encapsulate the underlying logging package:

Log4JLogger

Delegates to a Log4J Logger

Jdk14Logger

Wraps the standard logger that comes with the J2SE

LogKitLogger

Encapsulates the Avalon logkit logger

SimpleLogger

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

B.2. Apache Log4J

Apache Log4J, a powerful logging framework, is the de-facto standard in the Java Open Source community. Recently promoted from the Jakarta project, Log4J has done so well that it has been ported to C, C++, Perl, PHP, Python, .NET, and Ruby. Log4J provides the following features:

Configurable Destinations

Log4J logs messages to different output destinations, including files, JMS, and a DBMS.

Log Levels

Log4J has logging levels that enable you to configure which messages are logged to the output destination. For example, if debug-level logging is enabled, Log4J logs debug and higher level messages, and ignores trace-level messages.

Powerful Formatting

The formatting classes enable developers to specify the look and feel of logging messages.

Because a thorough discussion of Log4J is outside the scope of this book, please see the References section for complete documentation.

B.2.1. Log4J Core Concepts

Log4J's key classes and interfaces include:

Logger

Log4J's main class. `Logger` logs messages to an `Appender` based on the message's logging level.

Appender

An `Appender` sends logging messages to a particular destination. Here are some of the `Appender` interface's concrete implementations:

JDBCAppender

Stores logging messages in a database table.

FileAppender

Writes logging messages to a file.

DailyRollingFileAppender

Writes logging messages to a file that is rolled over at a user-defined interval. Most people let the log file roll over every 24 hours so you have a separate log file for each day.

JMSAppender

Publishes logging messages to a JMS Topic.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)[NEXT ▶](#)

B.3. Adding Application-Specific Properties to System Properties

The `InitServlet` uses `SystemPropertiesUtil` to add the `log4j.log.dir` property (that represents the directory that holds the log file) from the `log4j.extra.properties` to the System Properties. The `SystemPropertiesUtil` utility looks like [Example B-8](#).

Example B-8. SystemPropertiesUtil.java

```
package com.jbossatwork.util;

import java.util.*;

public class SystemPropertiesUtil {

    /**
     * Making the default (no arg) constructor private
     * ensures that this class cannot be instantiated.
     */
    private SystemPropertiesUtil( ) { }

    public static void addToSystemPropertiesFromPropsFile(String
propsFileName) {
        Properties props = ResourceLoader.getAsProperties(propsFileName),
        sysProps = System.getProperties( );

        // Add the keys/properties from the application-specific properties
        to
        // the System Properties.

        sysProps.putAll(props);
    }
}
```

The `SystemPropertiesUtil`'s `addToSystemPropertiesFromPropsFile()` method adds the keys/properties from an application-specific properties file to the System Properties. The application-specific properties file MUST reside in a directory listed on the `CLASSPATH`. The `SystemPropertiesUtil` uses the `ResourceLoader.getAsProperties()` method to retrieve application-specific properties file from the `CLASSPATH`. We'll cover the `ResourceLoader` utility after we show the `Log4jConfigurator`.

Now that we have the logging infrastructure in place, let's look more closely at the `Log4jConfigurator`.

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

B.4. Configuring Log4J with a Configuration File

The `InitServlet` uses the `Log4jConfigurator` to set up Log4J with our Log4J Configuration file, `jbossatwork-log4j.xml`. [Example B-9](#) shows the `Log4jConfigurator` class.

Example B-9. Log4jConfigurator.java

```
package com.jbossatwork.util;

import java.net.*;

import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.xml.DOMConfigurator;

public class Log4jConfigurator {

    /**
     * Making the default (no arg) constructor private
     * ensures that this class cannot be instantiated.
     */
    private Log4jConfigurator( ) { }

    /**
     * Configures Log4J for an application using the specified Log4J XML
     * configuration
     *
     * @param log4jXmlFileName The specified Log4J XML configuration file.
     */
    public static void setup(String log4jXmlFileName) {
        URL url = ResourceLoader.getAsUrl(log4jXmlFileName);

        if (url != null) {

            // An URL (from the CLASSPATH) that points to the Log4J XML
            // configuration file that was provided, so use Log4J's
            // DOMConfigurator with the URL to initialize Log4J with the
            // contents of the Log4J XML configuration file.

            DOMConfigurator.configure(url);
        } else {

            // An URL that points to the Log4J XML configuration file wasn't
            // provided, so use Log4J's BasicConfigurator to initialize Log4J.

            BasicConfigurator.configure( );
        }
    }
}
```

The `Log4jConfigurator.setup()` method configures Log4J for the JAW Motors application. The `ResourceLoader.getAsUrl()` method retrieves the Log4J configuration file from the `CLASSPATH`. If the configuration file is found, the `Log4jConfigurator.setup()` method uses the `Log4jDOMConfigurator.setup()` method to configure Log4J with our Log4J configuration file. Otherwise, the `Log4jConfigurator.setup()` method uses Log4J's `BasicConfigurator.configure()` method to configure Log4J with default settings.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

B.5. Loading Resources from the CLASSPATH

Both the `SystemPropertiesUtil` and `Log4jConfigurator` utilities use `ResourceLoader` to find resources (a property file and the Log4J configuration file) on the application `CLASSPATH`. The `ResourceLoader` utility looks like [Example B-10](#).

Example B-10. ResourceLoader.java

```
package com.jbossatwork.util;

import java.io.*;
import java.net.*;
import java.util.*;

public class ResourceLoader {

    /**
     * Making the default (no arg) constructor private
     * ensures that this class cannot be instantiated.
     */
    private ResourceLoader( ) { }

    public static Properties getAsProperties(String name) {
        Properties props = new Properties( );
        URL url = ResourceLoader.getAsUrl(name);

        if (url != null) {
            try {
                // Load the properties using the URL (from the CLASSPATH).

                props.load(url.openStream( ));
            } catch (IOException e) {
            }
        }
        return props;
    }

    public static URL getAsUrl(String name) {
        ClassLoader classLoader = Thread.currentThread(
        ).getContextClassLoader( );

        return classLoader.getResource(name);
    }
}
```

The `ResourceUtil.getAsProperties()` method calls `ResourceUtil.getAsUrl()` to find a properties file as an URL on the application `CLASSPATH`. The `props.load(url.openStream())` call first opens the `URL` as an `InputStream` and then loads the properties file.

The `ResourceUtil.getAsUrl()` method uses the Thread Context ClassLoader to find a properties file as an URL on the `CLASSPATH` application. The call to `Thread.currentThread().getContextClassLoader()` gets the current Thread's ClassLoader, and the `classLoader.getResource(propsFileName)` call searches for a properties filename on the application `CLASSPATH` and returns a `java.net.URL` that points to a resource (a properties file, data file, `.class` file, and so on). The Thread Context ClassLoader is the ClassLoader used by the creator of the Thread that runs your code. By using the Thread Context ClassLoader, we're guaranteed to load the resource as long as it's on the application's `CLASSPATH`. For more information on deployment , please see [Chapter 3](#). See [Appendix A](#) for more information on ClassLoaders.

Packaging properties files in a JAR is cleaner than using external property files that reside on a disk directory because the directory structure for your deployment environment could differ on each machine where you deploy your application. Using JAR files for packaging is a standard

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

B.6. Logging Deployment

The new files go into the Common JAR file, as follows:

- The root directory holds the following:
 - The Apache Jakarta Commons Logging JAR files
 - The Log4J JAR file
 - The `log4j.extra.properties` file that defines the property where the log file resides
 - The `commons-logging.properties` file that tells Jakarta Apache Commons Logging to use Log4J
 - The Log4J Configuration file, `jbossatwork-log4j.xml`
- Everything else remains unchanged.

The new `SystemPropertiesUtil`, `Log4jConfigurator`, and `ResourceLoader` objects are part of the `com.jbossatwork.util` package, so their class files reside in the Common JAR's `com/jbossatwork/util` directory. See [Chapter 3](#) for more information on the Common JAR file.

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

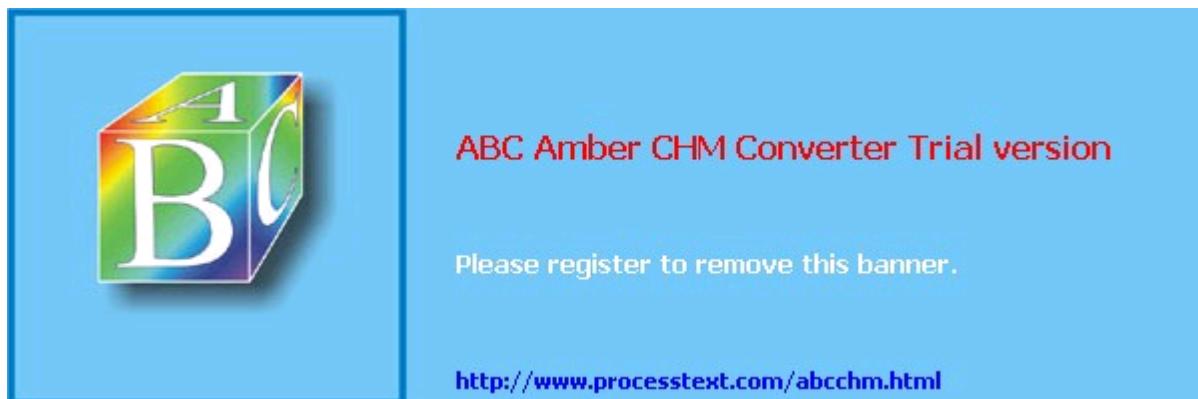
B.7. Logging Checklist

Before moving on to test the JAW Motors application's new logging functionality, let's recap what we've done to add logging to the application:

- Created the `commons-logging.properties` file that tells Apache Jakarta Commons Logging (JCL) to use Log4J as its implementation.
- Used Apache JCL calls in the JAW Motors application code.
- Added a Log4J Configuration file, `jbossatwork-log4j.xml`, to tell Log4J to format messages and where to log them.
- Developed a `log4j.extra.properties` file that defines where the log file resides.
- Wrote an Initialization Servlet to configure Log4J at JBoss startup, deferring the low-level setup details to utility classes.
- Deployed the logging implementation in the Common JAR:
- The Apache JCL and Log4J JARs, properties files, and new utility classes go in the root directory.

[◀ PREV](#)

[NEXT ▶](#)



 PREV

NEXT 

B.8. Testing Logging

Now that we've developed all the logging code and infrastructure, let's test the application to ensure that everything still works properly. Here are the steps to build and deploy the application:

- Type `ant` in the root directory of `appb` to build the project.
- Shut down JBoss so the Ant script can clean up the JBoss deployment area.
- Type `ant colddeploy` to deploy the EAR file (`jaw.ear`) to the `$JBOSS_HOME/server/default/deploy` directory. Notice that the Ant build script also creates the logfile directory (specified by `log4j.log.dir` in the `log4j.extra.properties` file).
- Start JBoss back up.

The Initialization Servlet runs at startup time, and you should see this message in the JAW Motors logfile, `${log4j.log.dir}/jbossatwork.log`:

```
2005-06-06 17:24:02,923 INFO [com.jbossatwork.InitServlet] - Testing Logging Setup  
...
```

Now point your browser to <http://localhost:8080/jaw>this takes you to the JAW Motors home page. After running a credit check and buying a car, the log messages in the `${log4j.log.dir}/jbossatwork.log` file should now look like this:

```
2005-06-06 17:24:02,923 INFO [com.jbossatwork.InitServlet] - Testing Logging Setup  
...  
2005-06-06 17:25:19,163 INFO [com.jbossatwork.ControllerServlet] - Credit Check:  
2005-06-06 17:25:19,163 INFO [com.jbossatwork.ControllerServlet] - Name = [Tom]  
2005-06-06 17:25:19,163 INFO [com.jbossatwork.ControllerServlet] - SSN = [[345834[958[34]]]  
2005-06-06 17:25:19,163 INFO [com.jbossatwork.ControllerServlet] - Email = [fred@acme.org]  
2005-06-06 17:25:19,293 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean] - CreditCheckProcessorBean.onMessage( ): Received message.  
2005-06-06 17:25:19,293 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean]-- Credit Check:  
2005-06-06 17:25:19,293 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean]--Name = [Tom]  
2005-06-06 17:25:19,293 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean] - SSN = [[345834[958[34]]]  
2005-06-06 17:25:19,293 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean]-- Email = [fred@acme.org]  
2005-06-06 17:25:19,303 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean]-- Verifying Credit ...  
2005-06-06 17:25:26,804 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean]-- Credit Check Result = [Pass Credit Check]  
2005-06-06 17:25:26,844 INFO [com.jbossatwork.ejb.CreditCheckProcessorBean]-- Sending Email to [fred@acme.org] ...  
2005-06-06 17:25:48,465 INFO [com.jbossatwork.ControllerServlet] - carId = [99],  
price = [13500.0]
```

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

B.9. Conclusion

In this appendix, we showed how to use Apache JCL as the high-level logging API used by an application. We hid the Log4J logging implementation to enable migration to another technology without changing the code. We then showed how to configure Apache JCL and Log4J in a J2EE/JBoss environment, and how to instrument your code with logging.

◀ PREV

NEXT ▶



◀ PREV

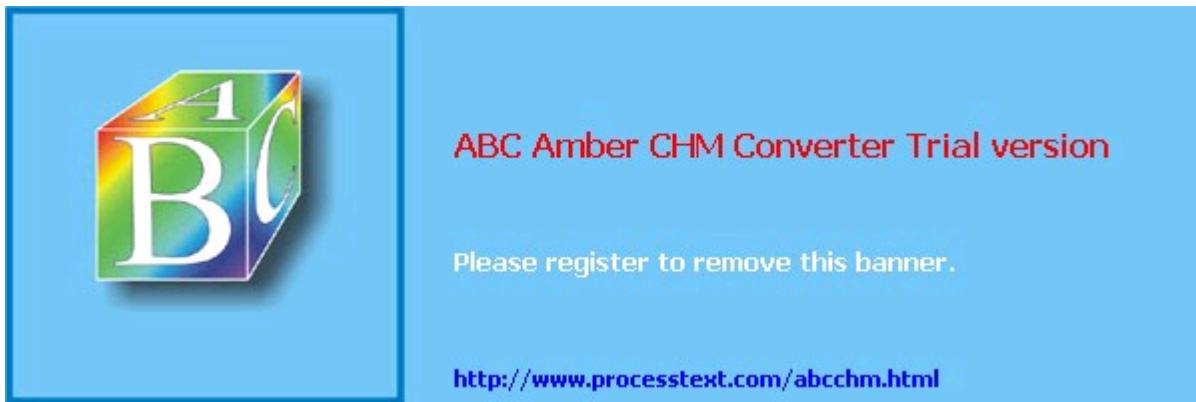
NEXT ▶

Appendix C. JAAS Tutorial

The Java Authentication & Authorization Service (JAAS) enables an application to protect its resources by restricting access to only users with proper credentials and permissions. JAAS provides a layer of abstraction between an application and its underlying security mechanisms, making it easier to change security technologies and realms without impacting the rest of the system.

◀ PREV

NEXT ▶



 PREV

NEXT 

C.1. JAAS

JAAS is a standard Java extension in J2SE 1.4, and it provides pluggable authentication that gives application designers a wide choice of security realms:

- DBMS
- Application Server
- LDAP
- Operating System (UNIX or Windows NT/2000)
- File System
- JNDI
- Biometrics

JAAS supports single sign-on for an application. Rather than forcing the user to log in to a web site, and then log in again to a forum or a backend legacy system used by the application, JAAS coordinates all these steps into one central login event to help coordinate access to all systems that the user needs.

We chose JAAS as the basis for our authentication strategy because:

- It provides a security context that covers the entire J2EE architecture from the web tier to the EJB tier.
- It is application-server neutral.
- It integrates with the Java 2 security model.
- It is part of the J2SE 1.4 extension API.
- It is more sophisticated than the other authentication mechanisms and provides more functionality.
- It supports single sign-on by coordinating multiple security realms.
- It addresses authorization in addition to authentication.
- It provides good encapsulation for authentication and authorization, enabling an application to be independent of the underlying security mechanisms it uses.

C.1.1. JAAS Core Concepts, Classes, and Interfaces

Here are the key pieces of the JAAS framework and the roles that they play:

`javax.security.auth.Subject`

A `Subject` is a user ("John Smith") or outside entity ("Acme") that accesses the system. The `Subject` groups one or more `Principals` together and holds the user's credentials.

`java.security.Principal`

Although not officially part of the JAAS packages (`javax.security.*`), the `Principal` acts as a role for a `Subject` (user).

Credentials

A credential can be a password, certificate, or key that identifies the user to the system. The JAW Motors application uses passwords.

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

C.2. Client-Side JAAS

To use JAAS from an external client, we need to take the following steps:

- Write the client code, using the `MyPassiveCallbackHandler` from the `CallbackHandler` section.
- Configure a client-side `LoginModule`.
- Set up a J2SE Security Policy file.

C.2.1. External Client Application that Uses JAAS

The following code snippet shows how an external application would use JAAS authentication, assuming that the user already entered his username and password. We first instantiate an application-specific `CallbackHandler` implementation, `MyPassiveCallbackHandler`, with the `userName` and `password`. We then create the JAAS `LoginContext` by using the application name along with our `CallbackHandler`. The "Client-JBossAtWorkAuth" application name comes from the `LoginModule` Configuration file see the "[Client-Side LoginModule Configuration](#)" section for details. The `LoginContext's login()` method then authenticates the user. If `login()` throws a `LoginException`, then the logon process failed. If the logon succeeds, the application calls the code the user is allowed to access, as in [Example C-3](#).

Example C-3. Sample external client

```
import javax.security.auth.login.*;
import javax.security.auth.*;
import java.security.*;

...
try {
    MyPassiveCallbackHandler myCallbackHandler = null;

    // Set Security Association CallbackHandler-specific settings
    myCallbackHandler = new MyPassiveCallbackHandler(userName, password);

    // Get Login Context (NOTE: Client-JBossAtWorkAuth is the application
    // name in the client-auth.conf LoginModule Configuration file)
    System.out.println("Creating the JAAS Login Context");
    LoginContext loginContext = new LoginContext("Client-JBossAtWorkAuth",
                                                myCallbackHandler);

    // Login
    System.out.println("Logging in as user [" + userName + "]");
    lc.login();

    // Protected code goes here.

} catch (LoginException le) {
    System.out.println(le.getMessage());
}
```

We've now written the core client code and the `CallbackHandler`, but there's still a little more work to do before we can run the client. We need to take the following steps:

- Configure a client-side `LoginModule`.
- Create a J2SE Security file.
- Set up the Client's `CLASSPATH`.

C.2.2. Client-Side LoginModule Configuration

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

◀ PREV

NEXT ▶

C.3. Conclusion

In this appendix, we reviewed portions of the Security chapter and added detailed coverage of other aspects of the JAAS API, including client-side `LoginModule` configuration and classes such as the `Callback`, `CallbackHandler`, and `LoginContext`.

◀ PREV

NEXT ▶



[◀ PREV](#)[NEXT ▶](#)

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of *JBoss at Work: A Practical Guide* is a golden eagle (*Aquila chrysaetos*), named for the golden feathers on the back of its neck. This large bird of prey is one of the two eagle breeds found in the United States, inhabiting parts of the West as well as Canada, Alaska, Eurasia, and northern Africa. The golden eagle makes its home in desert grasslands and above the timberline. There it can stretch its wings (7 feet across) and go for a nice swoop at 200 mph, or catch a rising mass of warm air called a thermal and spiral upward into the sky.

Golden eagles build large stick nests in trees or cliff walls. They may build multiple nests within a nesting range and alternate among them, depending on the year. Since golden eagles continually elaborate on their nests, the nests can grow quite large, reaching 8 to 10 feet across and 3 to 4 feet in depth. Both the male and female participate in the rearing of the eaglets, with the male doing most of the hunting and the female doing most of the incubating. If food is scarce, the larger of the eaglets may commit siblicide.

The young fledge when 72 to 84 days old and depend upon their parents for another 3 months. After this period they either migrate or move out of their parents' territory, but they generally winter in their natal area. At four years of age, golden eagles mate. They often stay paired with the same mate for lifeabout 30 years. Golden eagles are excellent hunters and for this reason are rarely forced to migrate far from their nesting territory. They feast on over 50 species of mammals, 48 birds, 5 reptiles, and 2 fish. Among these are included prairie dogs, rabbits, ground squirrels, grouse, ducks, chukars, marmots, foxes, skunks, cats, meadowlarks, and snakes. Golden eagles are protected in the U.S. through the U.S. Fish and Wildlife Service. Possessing a feather or any other body part belonging to this bird will incur a \$10,000 fine or a jail term of up to 10 years. (There are some exceptions for Native American traditions.)

Colleen Gorman was the production editor and proofreader, and Ann Schirmer was the copyeditor for *JBoss at Work: A Practical Guide*. Jamie Peppard and Genevieve d'Entremont provided quality control. Lorannah Dimant provided production assistance. Johnna VanHoose Dinse wrote the index.

Karen Montgomery designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from Johnson's Natural History. Karen Montgomery produced the cover layout with Adobe InDesign CS using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Joe Wizda to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birk; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano, Jessamyn Read, and Lesley Borash using Macromedia FreeHand MX and Adobe Photoshop CS. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Lydia Onofrei.

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[O\]](#) [\[P\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

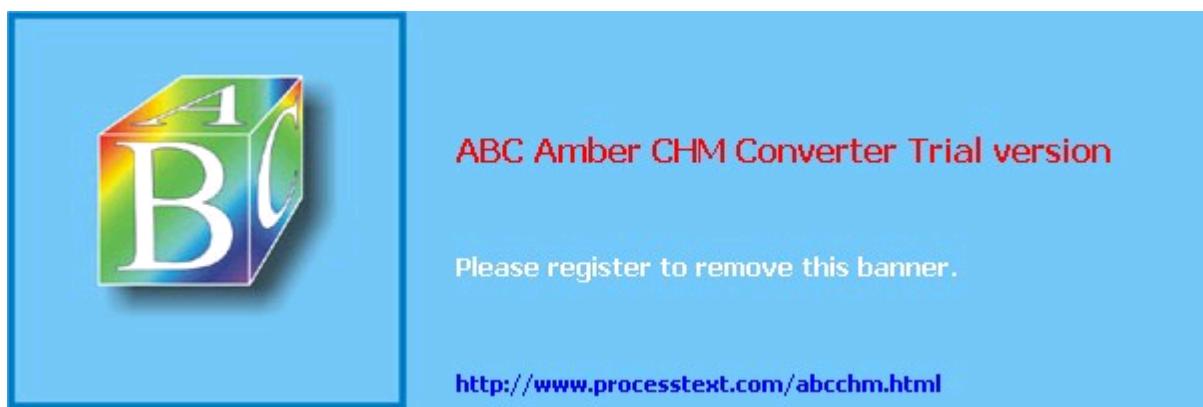
Index

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[O\]](#) [\[P\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[<attribute name> element, hibernate-service.xml file](#)
[<class> element, car.hbm.xml file](#)
[<id> element, car.hbm.xml file](#)
[<mbean> element, hibernate-service.xml file](#)
[<property> element, car.hbm.xml file](#)

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[ACCOUNTING table](#)

[AccountingDAO, HibernateAccountingDAO](#)

[AccountingDTO object](#)

Ant

[?? automated deployment and](#)

[?? build script, XDoclet and](#)

[?? databases and](#)

[?? FAR task](#)

[?? EJB JAR file creation](#)

[?? HBM file](#)

[?? installation](#)

[?? WAR and](#)

Apache

[?? Axis, Web Services client and](#)

[?? Commons Logging](#)

[?? JCL API](#)

[?? Log4J](#)

[???? Appender interface](#)

[???? Common Jar file](#)

[???? configuration](#)

[???? configuration file](#)

[???? initialization](#)

[???? Layout interface](#)

[???? Logger class](#)

[???? PatternLayout](#)

[???? resource loading](#)

[Appender interface \(Log4J\)](#)

[application-specific properties, System properties and](#)

[application.xml file](#)

applications

[?? presentation tier](#)

[?? three-tier](#)

architecture, JMS

[archives, HARs \(Hibernate Archives\)](#)

[asynchronous processing, EJBs and](#)

authentication

[?? declarative, web.xml and](#)

[?? form-based](#)

[???? login form](#)

[?? J2EE security](#)

[?? web-based](#)

authorization

[?? declarative, web.xml and](#)

[?? J2EE security](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[Bean Class](#)

[bin/ directory](#)

[BMPs \(Bean-Managed Persistence\)](#)

[BMTs \(Bean-Managed Transactions\)](#)

[??_MDBs and](#)

[build.xml files](#)

[??_common.jar and](#)

[business logic, removing from Controller Servlet](#)

[Business tier](#)

[Buy Car page, JAW Motors example](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[C#, Web Services client](#)
[caching, EJBs and](#)
[Callback interface \(JAAS\)](#)
[CallbackHandler interface \(JAAS\)](#)
[car.hbm.xml file](#)
[CarBean](#)
[CarDAO](#)
[CarDTO, status field](#)
classes
[?? Bean Class](#)
[?? JAAS](#)
[?? JMS API](#)
[ClassLoader Repository](#)
ClassLoaders
[?? debugging issues](#)
[?? Delegation Model](#)
[?? Extension ClassLoader](#)
[?? J2EE](#)
[?? JBoss](#)
[?? namespaces](#)
[?? options](#)
[?? troubleshooting](#)
[client/ directory](#)
clients
[?? Session Beans and](#)
[?? Web Services](#)
[???? implementation](#)
[CMPs \(Container-Managed Persistence\)](#)
[CMTs \(Container-Managed Transactions\) 2nd](#)
[?? Hibernate and](#)
[?? MDBs and](#)
[collections, Web Services and](#)
[common directory](#)
[Common JAR](#)
[common sub-project](#)
[commons-logging.properties file](#)
[complexity, EJBs and](#)
configuration
[?? Log4J \(Apache\)](#)
[?? Log4J files](#)
[?? LoginModule](#)
[?? servers](#)
[???? directory structure](#)
[Context Root, WAR files](#)
Controller Servlet
[?? business logic removal](#)
[?? refactoring](#)
[?? Session Beans, calling](#)
controllers
[Credit Check link](#)
[credit check, JAW Motors example](#)
[CSS \(cascading style sheets\), WAR files](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[DAO \(Data Access Object\). creating](#)
[data types. Web Services](#)
[database connection pooling. EJBs and](#)
[databases](#)
?? [Ant and](#)
?? [checklist](#)
[DataSource](#)
?? [descriptors](#)
?? [Hypersonic database and](#)
?? [JDBC](#)
?? [JDBC driver JARs](#)
[debug logging level](#)
[debugging](#)
?? [ClassLoaders and](#)
?? [logging and](#)
[declarative authentication. web.xml and](#)
[declarative authorization. web.xml](#)
[declarative security. J2EE](#)
?? [administrative actions](#)
[declarative transactions. EJBs and](#)
[deployment](#)
?? [as WAR file](#)
?? [automated. Ant and](#)
?? [EAR file](#)
?? [EARs and](#)
?? [EJB JAR file](#)
?? [JavaMail API](#)
?? [JMS Destinations](#)
?? [MDBs](#)
???? [XDoclet and](#)
?? [Stateless Session beans. automating with XDoclet](#)
?? [WARs and](#)
?? [Web Services](#)
???? [Ant script](#)
???? [automation](#)
???? [testing](#)
???? [XDoclet and](#)
?? [web-based. EJB-based JNDI references](#)
[deployment descriptors](#)
?? [EJB](#)
???? [JavaMail-based JNDI references](#)
?? [JMS-based JNDI references](#)
[developers](#)
[directories](#)
?? [bin/](#)
?? [client/](#)
?? [common](#)
?? [docs/](#)
?? [dtd/](#)
?? [examples/](#)
?? [lib/](#)
?? [licenses/](#)
?? [schema/](#)
?? [server configuration](#)
?? [server/ 2nd](#)
?? [structure](#)
?? [temporary](#)
?? [tests/](#)
?? [Web Services client](#)
?? [webapp](#)
[distributed transactions. EJBs and](#)
[docs/ directory](#)
[domains](#)
?? [mid-level](#)
?? [top-level](#)
[drivers. JDBC](#)
[dtd/ directory](#)

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

EAR files

?? [Ant EAR task](#)

?? [application.xml file](#)

?? [Common JAR](#)

?? [deployment](#)

?? [EJB JAR file, adding](#)

?? [EJB module](#)

?? [HAR, adding](#)

?? [Java module](#)

?? [WAR file comparison](#)

?? [Web module](#)

[EARs \(Enterprise Archives\)](#)

[EJB deployment descriptors, JavaMail-based JNDI references](#)

EJB JAR file

?? [creating, Ant task and](#)

?? [FAR and](#)

[EJB module, FAR](#)

[EJB tier security](#)

?? [web tier security integration](#)

[EJB-based programmatic security](#)

[EJBs \(Enterprise JavaBeans\)](#)

?? [asynchronous processing](#)

?? [caching and](#)

?? [complexity and](#)

?? [database connection pooling and](#)

?? [declarative transactions](#)

?? [deployment descriptors](#)

?? [distributed transactions and](#)

?? [ejb sub-project](#)

?? [ejb-jar.xml and](#)

?? [Entity Beans](#)

?? [JAR files](#)

?? [JNDI references, web-based deployment descriptors](#)

?? [local calls](#)

?? [MDBs \(Message-Driven Beans\)](#)

?? [persistence and](#)

?? [read-only applications and](#)

?? [remote access and](#)

?? [remote calls](#)

?? [security](#)

?? [Session Beans](#)

???? [Stateful](#)

???? [Stateless](#)

?? [settings automation, XDoclet and](#)

?? [stateless session beans](#)

?? [testing secure methods](#)

?? [threads and](#)

?? [transactions](#)

???? [BMTs](#)

???? [CMTs](#)

???? [Mandatory](#)

???? [Never](#)

???? [NotSupported](#)

???? [Required](#)

???? [RequiresNew](#)

???? [Supports](#)

email

?? [sending messages](#)

???? [JavaMail](#)

???? [JavaMail API](#)

???? [MDB](#)

?? [TextEmail utility](#)

[ENC \(Environmental Naming Context\)](#)

[Entity Beans](#)

[error logging level](#)

[examples/ directory](#)

[Extension ClassLoader](#)

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[fatal logging level](#)

[filterByStatus\(\).HibernateDAO](#)

[form-based authentication](#)

[??_login form](#)

[forms.login](#)

[FQDN \(Fully Qualified Domain Name\)](#)

[framew orks](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

HARs (Hibernate Archives)

[??_creating](#)

[??_EAR and](#)

[HBM files](#)

[??_Ant tasks](#)

[Hibernate](#)

[??_checklist](#)

[??_CMT and](#)

[??_introduction](#)

[??_MBean service, configuration file](#)

[hibernate-service.xml file](#)

[HibernateAccountingDAO](#)

[HibernateCarDAO](#)

[??cars](#)

[????_adding](#)

[????_deleting](#)

[????_editing](#)

[HibernateDAO.filterByStatus\(\)](#)

Home Interfaces

[??_Local Home Interface](#)

[??_Remote Home Interface](#)

[hot deployment](#)

Hypersonic databases

[??_DataSource and](#)

[??_instances](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

implementation

[?? J2EE Web Services](#)

[?? Web Services client](#)

[info logging level](#)

initialization

[?? Log4J \(Apache\)](#)

[?? Log4J Servlet](#)

installation

[?? Ant](#)

[?? Java](#)

[?? JBoss](#)

[?? JDK](#)

[?? XDoclet](#)

interfaces

[?? Home Interfaces](#)

[???? Local Home Interface](#)

[???? Remote Home Interface](#)

[?? JAAS](#)

[?? Local Interface](#)

[?? Remote Interface](#)

[InventoryFacadeBean EJB, modifying](#)

[InventoryFacadeBean, buyCar\(\)](#)

Iteration 1, JAW Motors example

[?? review](#)

[?? Session Bean introduction](#)

[?? testing](#)

[Iteration 2, JAW Motors example](#)

[?? review ing](#)

[?? testing](#)

[Iteration 3, JAW Motors example](#)

[?? review ing](#)

[?? testing](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

J2EE

?? [ClassLoaders and](#)

?? Declarative Security

???? [administrative actions](#)

?? [security](#)

???? [authentication](#)

???? [authorization](#)

???? [declarative](#)

???? [guests](#)

???? [managers](#)

?? [Web Services and](#)

???? [implementation](#)

[JAAS \(Java Authentication & Authorization Service\)](#)

?? [Callback interface](#)

?? [CallbackHandler interface](#)

?? [classes](#)

?? [client-side](#)

?? [domain settings, jboss.xml](#)

?? [domain, jboss.xml](#)

?? [interfaces](#)

?? [jboss-web.xml, domain settings](#)

?? [LoginContext](#)

?? [LoginModule](#)

?? [LoginModule Configuration file](#)

?? [security realm deployment](#)

?? [tutorial](#)

JAR files

?? [EJB JAR files](#)

?? [JDBC drivers](#)

Java installation

Java module, EAR

JavaMail API

?? [deployment](#)

?? [introduction](#)

?? [JNDI calls and](#)

?? JNDI references

???? [EJB deployment descriptors and](#)

???? [XDoclet and](#)

?? [POP \(Post Office Protocol\) and](#)

?? [sending messages 2nd](#)

?? [sessions](#)

?? [SMTP \(Simple Mail Transfer Protocol\) and](#)

JAW Motors example

?? [Buy Car page](#)

?? [credit check, JavaMail and](#)

?? [introduction](#)

?? Iteration 1

???? [review](#)

???? [Session Bean introduction](#)

???? [testing](#)

?? [Iteration 2](#)

???? [review](#)

???? [testing](#)

?? [Iteration 3](#)

???? [reviewing](#)

???? [testing](#)

?? Session Beans

???? [adding](#)

???? [calling from Controller Servlet](#)

???? [introduction](#)

JAX-RPC mapping file

?? [Web Services](#)

?? [Web Services client](#)

JBoss

?? [ClassLoaders](#)

?? [directory structure](#)

?? [installation](#)

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[Layout interface, Log4J \(Apache\)](#)

[lib/ directory](#)

[licenses/ directory](#)

[local calls, EJBs](#)

[Local Home Interface](#)

[Local Interface](#)

[Log interface, JCL API](#)

[Log4J \(Apache\)](#)

[??_Appender interface](#)

[??_configuration](#)

[????_resource loading](#)

[??_configuration file](#)

[??_deployment](#)

[??_initialization](#)

[????_Servlet](#)

[??_Layout interface](#)

[??_Logger class](#)

[??_PatternLayout](#)

[log4j.extra.properties file](#)

[Log4Logger](#)

[Logger class, Log4J](#)

[logging](#)

[??_auditing](#)

[??_checklist](#)

[??_Common Jar file](#)

[??_Commons Logging package, commons-logging.properties file](#)

[??_debug level](#)

[??_debugging and](#)

[??_error level](#)

[??_fatal level](#)

[??_info level](#)

[??_JCL API](#)

[??_Jdk14Logger](#)

[??_Log4J \(Apache\)](#)

[??_Log4Logger](#)

[??_logging levels](#)

[??_LogKitLogger](#)

[??_review ing](#)

[??_SimpleLogger](#)

[??_testing](#)

[??_trace logging level](#)

[??_w arn level](#)

[login form](#)

[LoginContext \(JAAS\)](#)

[LoginModule \(JAAS\) 2nd](#)

[??_configuration](#)

[??_security realm deployment](#)

[LoginModule Configuration file \(JAAS\)](#)

[LogKitLogger](#)

[lookups, JNDI](#)

[??_creating](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[master build file](#)

[MBean service, Hibernate configuration files](#)

[MDBs \(Message-Driven Beans\) 2nd 3rd](#)

[??_BMTs and](#)

[??_deployment](#)

[????_XDoclet and](#)

[??_sending_email_messages](#)

[??_transactions](#)

[??_writing](#)

[message models \(JMS\)](#)

[??_P2P](#)

[??_Pub-Sub](#)

[messages \(JMS\), sending](#)

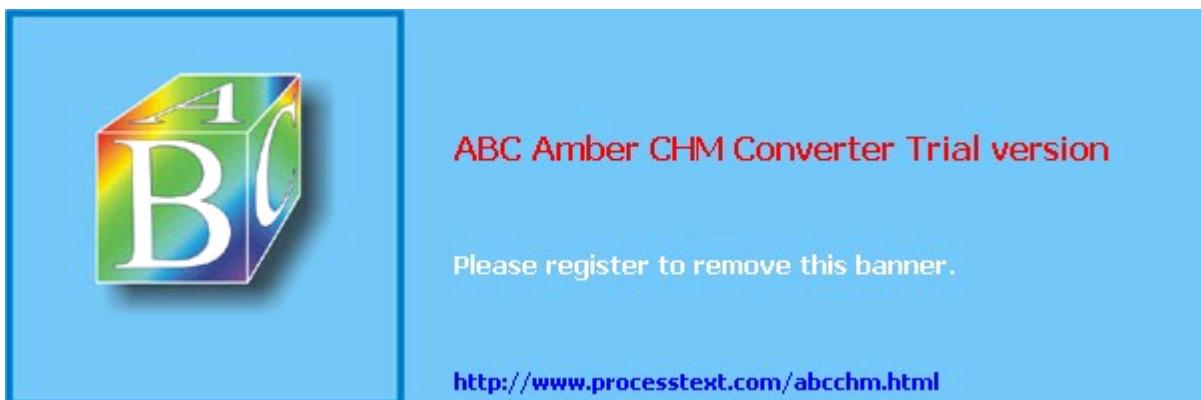
[MLDs \(mid-level domains\)](#)

[MVC \(Model/View/Controller\) pattern](#)

[MVC framework model](#)

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

Index

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[O\]](#) [\[P\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[OODBMS \(object-oriented databases\)](#)

[ORMs \(Object/Relational Mappers\)](#)

[??_cons](#)

[??_pros](#)

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[P2P \(Point-to-Point\) messaging model \(JMS\)](#)

[PatternLayout \(Log4J\)](#)

[Perl, Web Services client](#)

persistence

[??_EJBs and](#)

[??_options](#)

[Persistence tier](#)

[??_definition](#)

[POJO \(Plain Old Java Object\)](#)

[POP \(Post Office Protocol\), JavaMail and](#)

[presentation tier](#)

[programmatic security, EJB-based](#)

[programmatic web-based security](#)

[programmers](#)

[properties, application-specific, system properties and](#)

[Pub-Sub \(Publish-Subscribe\) message model, JMS](#)

[Python, Web Services client](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

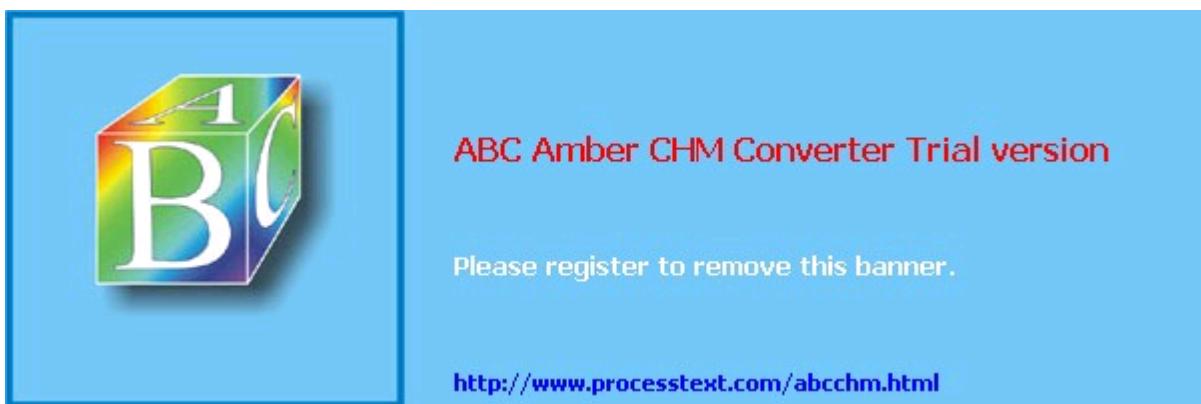
Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[read-only applications, EJBs and](#)
[refactoring, Controller Servlet](#)
[remote access, EJBs and](#)
[remote calls, EJBs](#)
[Remote Home Interface](#)
[Remote Interface](#)
[resource loading, Log4J](#)

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)[NEXT ▶](#)

Index

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[O\]](#) [\[P\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[schema/ directory](#)

security

[??_access restriction](#)

[??_EJB tier](#)

[??_EJBs and](#)

[??_J2EE](#)

[????_authentication](#)

[????_authorization](#)

[????_declarative](#)

[????_guests](#)

[????_managers](#)

[??_JAAS](#)

[??programmatic](#)

[????_EJB-based](#)

[????_web-based](#)

[??_web_testing](#)

[??_web-based](#)

[??_web-based authentication](#)

security credentials propagation

[??_web tier](#)

[??_web.xml](#)

[security realm](#)

[??deployment](#)

[????_JAAS-based](#)

[????_LoginModule](#)

[SEI \(Service Endpoint Interface\), Web Services 2nd](#)

[??_ejb-jar.xml modification](#)

[??_webservices.xml](#)

[server configuration](#)

[??_directory structure](#)

[server/ directory 2nd](#)

[Service](#)

[Service locator, JNDI lookup calls](#)

[servlet container](#)

[Session Beans](#)

[??_clients](#)

[??_JAW Motors example](#)

[????_adding](#)

[????_calling from Controller Servlet](#)

[??_local interfaces and](#)

[??_Stateful](#)

[??_Stateless](#)

[SimpleLogger](#)

[SMTP \(Simple Mail Transfer Protocol\), JavaMail and](#)

[SOAP \(Simple Object Access Protocol\)](#)

[SOAP Servlet](#)

[SSL configuration](#)

[Stateful Session Beans](#)

[Stateless Session Beans](#)

[??_deployment automation, XDoclet and](#)

[Sun Microsystems, Login Modules](#)

[system properties, application-specific properties and](#)

[◀ PREV](#)[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[Tagish LoginModule](#)

[taglibs](#)

[temporary directories](#)

[testing](#)

[??_EJB secure methods](#)

[??_Iteration 1, JAW Motors example](#)

[??_Iteration 2, JAW Motors example](#)

[??_Iteration 3, JAW Motors example](#)

[??_logging](#)

[??_secure JSPs](#)

[??_web security](#)

[??_Web Services client](#)

[??_Web Services deployment](#)

[tests/ directory](#)

[TextEmail utility](#)

[threads, EJBs and](#)

[three-tier applications](#)

[tiers](#)

[??_Business](#)

[??_Persistence](#)

[??_Web tier](#)

[TLDs \(top-level domains\)](#)

[tools setup](#)

[top-level domains \(TLDs\)](#)

[trace logging level](#)

[transactions](#)

[??_declarative, EJBs and](#)

[??_distributed, EJBs and](#)

[??EJBs](#)

[????_BMTs](#)

[????_CMTs](#)

[????_Mandatory](#)

[????_Never](#)

[????_NotSupported](#)

[????_Required](#)

[????_RequiresNew](#)

[????_Supports](#)

[??_MDBs](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[O\]](#) [\[P\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[Unified API \(JMS\)](#)

[URLs, Web Services](#)

[◀ PREV](#)

[NEXT ▶](#)



[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[VBAD \(Virtual Big Auto Dealership\)](#)

[View Cars page](#)

[??CSS](#)

[??HTML](#)

[??JSP](#)

[??JSTL](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

 PREV

NEXT 

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

WAR files

?? [Ant and](#)

?? [Context Root](#)

?? [CSS](#)

?? [deployment as](#)

?? [EAR comparison](#)

?? [HTML](#)

?? [JSP](#)

?? [JSTL](#)

[warn logging level](#)

[WARs \(Web Archives\)](#)

?? [Common Jar](#)

[Web module, EAR](#)

[web security testing](#)

[Web Service Proxy](#)

Web Services

?? [architecture](#)

?? [checklist](#)

?? [client](#)

?? [checklist](#)

?? [implementation](#)

?? [testing](#)

?? [collections and](#)

?? [data types](#)

?? [deployment](#)

?? [Ant script](#)

?? [automation](#)

?? [testing](#)

?? [XDoclet and](#)

?? [InventoryFacadeBean EJB modification](#)

?? [J2EE and](#)

?? [implementation](#)

?? [JAX-RPC Mapping file](#)

?? [JBoss 4.x and](#)

?? [JWSDP and](#)

?? [SEI 2nd](#)

?? [ejb-jar.xml](#)

?? [webservices.xml](#)

?? [SOAP Servlet 2nd](#)

?? [URLs](#)

?? [VBAD and](#)

?? [Web Service Proxy](#)

?? [WSDL file 2nd](#)

[Web tier](#)

?? [EJB tier security integration](#)

[web-based authentication](#)

web-based deployment descriptors

?? [EJB-based JNDI references](#)

?? [JNDI references, JMS-based](#)

[web-based security](#)

?? [programmatic](#)

[web.xml](#)

?? [access restriction](#)

?? [authentication, declarative](#)

?? [authorization, declarative](#)

?? [jboss-web.xml, JAAS domain settings](#)

?? [JNDI references](#)

?? [security credential propagation](#)

[webapp directory](#)

[webapp sub-project](#)

[WSDL \(Web Services Definition Language\)](#)

?? [files](#)

◀ PREV

NEXT ▶



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

[NEXT ▶](#)

Index

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[H](#)] [[I](#)] [[J](#)] [[L](#)] [[M](#)] [[O](#)] [[P](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)]

[XDoclet](#)

[?? Ant build script](#)

[?? EJB security automation](#)

[?? installation](#)

[?? JavaMail-based JNDI references](#)

[?? MDB deployment](#)

[?? Servlet deployment](#)

[?? Stateless Session Beans deployment automation](#)

[?? Web Services deployment and](#)

[XML application xml file](#)

[◀ PREV](#)

[NEXT ▶](#)



ABC Amber CHM Converter Trial version

Please register to remove this banner.

<http://www.processtext.com/abcchm.html>

[◀ PREV](#)

Index

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[L\]](#) [\[M\]](#) [\[O\]](#) [\[P\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[YAGNI \(You Ain't Gonna Need It\)](#)

[◀ PREV](#)

