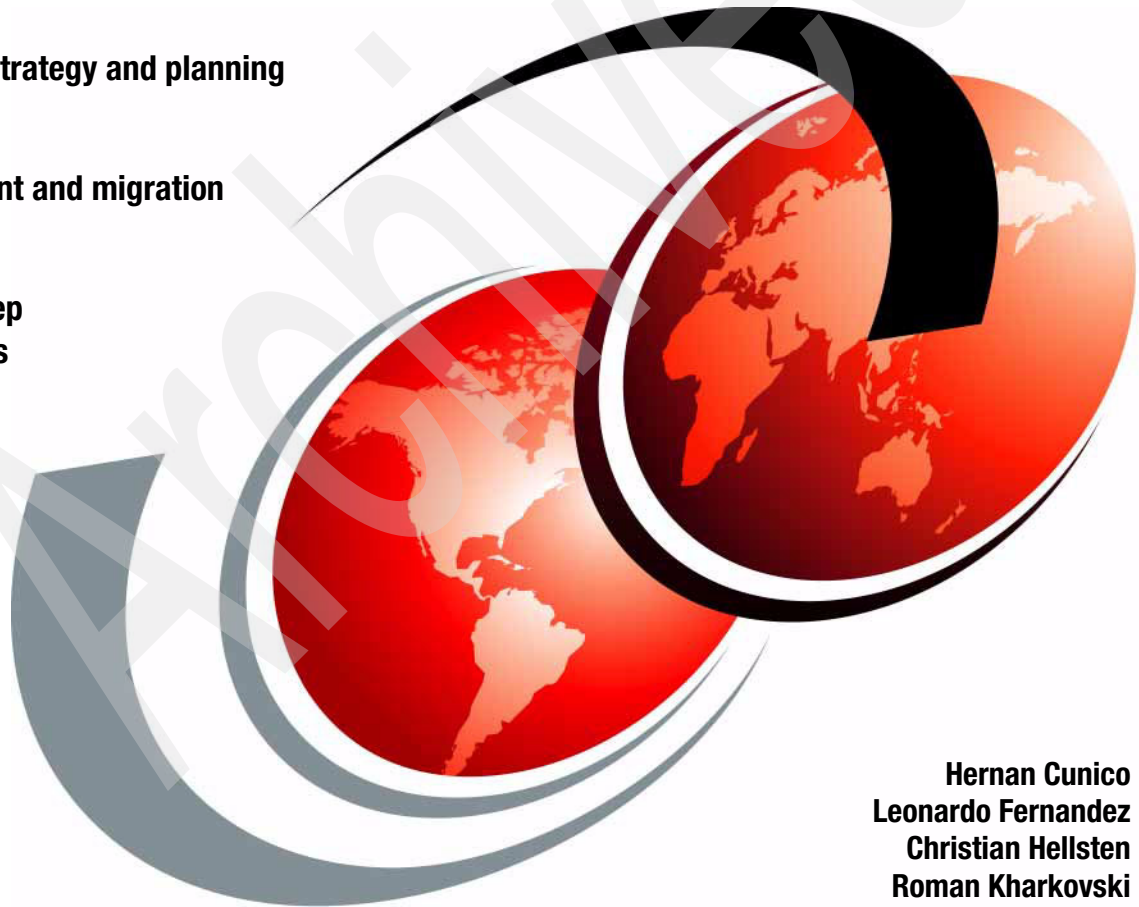


# Migrating Applications from WebLogic, JBoss and Tomcat to WebSphere V6

Migration strategy and planning

Development and migration tools

Step-by-step instructions



Hernan Cunico  
Leonardo Fernandez  
Christian Hellsten  
Roman Kharkovski





International Technical Support Organization

## **Migrating Applications from WebLogic, JBoss and Tomcat to WebSphere V6**

August 2005

Archived

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

## **First Edition (August 2005)**

This edition applies to Version 6, Release 0, Modification 0 of WebSphere Application Server.

© Copyright International Business Machines Corporation 2005. All rights reserved.

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

# Contents

<b>Notices</b> .....	ix
Trademarks .....	x
 <b>Preface</b> .....	xi
The team that wrote this redbook .....	xi
Become a published author .....	xiii
Comments welcome .....	xiii
 <b>Chapter 1. Introduction</b> .....	1
1.1 Our objective .....	2
1.2 Scope of this book .....	2
1.3 Applications covered in this book .....	3
1.4 What is not covered in this book .....	5
1.5 How to use this book .....	6
 <b>Chapter 2. WebSphere overview</b> .....	9
2.1 About WebSphere .....	10
2.2 WebSphere Application Server .....	11
2.2.1 WebSphere Application Server V6 - Express .....	12
2.2.2 WebSphere Application Server V6 - Base .....	12
2.2.3 WebSphere Application Server V6 - Network Deployment .....	13
2.2.4 WebSphere Extended Deployment .....	15
2.2.5 Supported platforms and software .....	19
2.3 Development and deployment tools .....	22
2.3.1 Rational Application Developer V6 .....	22
2.3.2 Rational Web Developer V6 .....	24
2.3.3 Application Server Toolkit (ASTk) .....	26
2.3.4 WebSphere Rapid Deployment .....	26
2.4 Performance and analysis tools .....	27
2.4.1 Memory analysis .....	28
2.4.2 Thread analysis .....	28
2.4.3 Execution time analysis .....	29
2.4.4 Code coverage .....	29
2.4.5 Probe kit .....	29
 <b>Chapter 3. Migration strategy and planning</b> .....	31
3.1 Migration strategy considerations .....	32
3.1.1 Getting help .....	32
3.1.2 Migrating the different environments .....	32

3.1.3 Strategies to handle migration complexity . . . . .	33
3.2 Application servers interoperability . . . . .	36
3.3 Migration planning activities . . . . .	45
3.3.1 Gathering concerned parties together. . . . .	46
3.3.2 Evaluating current assets . . . . .	46
3.3.3 Assessing the high-level application architecture . . . . .	47
3.3.4 Reviewing and validating the application code . . . . .	47
3.3.5 Reviewing the development environment . . . . .	49
3.3.6 Reviewing the runtime environment . . . . .	50
3.3.7 Reviewing the current build and deployment processes. . . . .	50
3.3.8 Assessing current skills. . . . .	50
3.3.9 Reviewing time constraints . . . . .	51
3.3.10 Creating a detailed migration plan . . . . .	51
3.4 Migration alternatives . . . . .	51
3.4.1 Alternative 1: Keeping the existing development environment . . . . .	52
3.4.2 Alternative 2: Migrating XDoclet tags to WRD . . . . .	53
3.4.3 Alternative 3: Using XDoclet support for WebSphere . . . . .	55
3.4.4 Alternative 4: Developing using Rational Application Developer V6 . . . . .	56
3.5 Migrating the site . . . . .	58
3.5.1 Switchover migration. . . . .	58
3.5.2 Coexistence migration. . . . .	58
3.5.3 Offline migration . . . . .	58
3.5.4 Database migration . . . . .	59
3.6 Testing the migrated site . . . . .	59
3.7 Going live . . . . .	60
3.8 Naming conventions . . . . .	60
<b>Chapter 4. Installation and configuration . . . . .</b>	<b>63</b>
4.1 Introduction . . . . .	64
4.2 Hardware and software installed. . . . .	64
4.3 Rational Application Developer V6 . . . . .	64
4.3.1 Installing the interim fix . . . . .	66
4.3.2 Testing the application server . . . . .	67
4.4 WebSphere Application Server V6 . . . . .	70
4.4.1 Installing the latest fixpack . . . . .	72
4.4.2 WebSphere profiles. . . . .	73
4.4.3 Creating new WebSphere profiles . . . . .	75
4.5 WebSphere Administrative Console . . . . .	78
4.5.1 Managing the application servers . . . . .	79
4.5.2 Universal Test Client . . . . .	81
4.6 Administrative scripting . . . . .	83
4.6.1 Examples . . . . .	84
4.7 DB2 Universal Database V8.2. . . . .	86

4.8 Installation directories . . . . .	87
<b>Chapter 5. Common migration issues.</b> . . . . .	89
5.1 J2EE application server compability . . . . .	90
5.1.1 Differences in J2EE implementations . . . . .	90
5.1.2 Using vendor-specific features . . . . .	91
5.1.3 Class loader related problems . . . . .	92
5.1.4 Deployment descriptors . . . . .	97
5.2 Application portability . . . . .	99
5.2.1 Application packaging . . . . .	99
5.2.2 Use of native code . . . . .	104
5.2.3 Database-related issues . . . . .	104
5.2.4 JMS . . . . .	106
5.2.5 JNDI . . . . .	106
5.2.6 J2EE application clients . . . . .	108
5.3 J2EE 1.3 to 1.4 migration considerations . . . . .	109
5.3.1 Enterprise Java Beans . . . . .	110
5.3.2 Java Server Pages . . . . .	110
5.3.3 Servlets . . . . .	110
<b>Chapter 6. Migrating from BEA WebLogic</b> . . . . .	111
6.1 Introduction . . . . .	112
6.2 Prerequisites and assumptions . . . . .	112
6.3 BEA WebLogic Server 8.1 installation . . . . .	114
6.4 Trade for BEA WebLogic Server 8.1 migration . . . . .	115
6.4.1 Migration approach . . . . .	116
6.4.2 Configuring the initial environment . . . . .	117
6.4.3 Migrating the sample application . . . . .	125
6.4.4 Summary . . . . .	144
6.5 xPetstore EJB migration . . . . .	144
6.5.1 Migration approach . . . . .	145
6.5.2 Configuring the initial environment . . . . .	146
6.5.3 Migrating the sample application . . . . .	153
6.5.4 Summary . . . . .	172
<b>Chapter 7. Migrating from JBoss</b> . . . . .	175
7.1 Introduction . . . . .	176
7.2 Prerequisites and assumptions . . . . .	177
7.3 Software installation . . . . .	178
7.3.1 JBoss . . . . .	178
7.3.2 Apache Ant . . . . .	179
7.4 xPetstore EJB migration using Alternative 2 . . . . .	179
7.4.1 Migration approach . . . . .	180
7.4.2 Configuring the initial environment . . . . .	181

7.4.3 Migrating the application . . . . .	191
7.5 xPetstore EJB migration using Alternative 3 . . . . .	228
7.5.1 Migration approach . . . . .	229
7.5.2 Configuring the initial environment . . . . .	229
7.5.3 Migrating the xPetstore EJB application . . . . .	230
<b>Chapter 8. Migrating from Tomcat . . . . .</b>	<b>257</b>
8.1 Introduction . . . . .	258
8.2 Prerequisites and assumptions . . . . .	258
8.3 Software installation . . . . .	260
8.3.1 Apache Tomcat . . . . .	260
8.3.2 Apache Maven . . . . .	261
8.3.3 Apache Ant . . . . .	261
8.3.4 hMailServer . . . . .	262
8.4 ivata groupware migration . . . . .	262
8.4.1 Migration approach . . . . .	263
8.4.2 Configuring the source environment . . . . .	263
8.4.3 Migrating the ivata groupware application . . . . .	267
8.5 xPetstore Servlet migration . . . . .	280
8.5.1 Migration approach . . . . .	281
8.5.2 Configuring the source environment . . . . .	281
8.5.3 Migrating the xPetstore Servlet application . . . . .	286
<b>Appendix A. Development tips for portable applications . . . . .</b>	<b>295</b>
End-to-end life cycle . . . . .	296
Software development life cycle key concepts . . . . .	296
The Rational Unified Process . . . . .	297
General development best practices . . . . .	300
Iterative development . . . . .	300
Requirements definition . . . . .	302
Object Oriented (OO) approach to design and programming . . . . .	303
Modeling languages . . . . .	304
Regular reviews and check points . . . . .	305
Java development best practices . . . . .	305
Code documentation . . . . .	306
Unit testing . . . . .	307
Automation . . . . .	310
Enterprise Java development best practices . . . . .	312
Modularization . . . . .	312
Follow the standards . . . . .	313
Design patterns . . . . .	314
J2EE best practices . . . . .	318
<b>Appendix B. Additional material . . . . .</b>	<b>323</b>



Locating the Web material .....	323
Using the Web material .....	324
System requirements for downloading the Web material .....	324
How to use the Web material .....	324
<b>Related publications</b> .....	325
IBM Redbooks .....	325
Online resources .....	325
How to get IBM Redbooks .....	328
Help from IBM .....	328
<b>Index</b> .....	329



# Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:  
*IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.*

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

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

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

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

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

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

## COPYRIGHT LICENSE:

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

# Trademarks

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

@server®  
e-server®  
Redbooks (logo) ™  
developerWorks®  
e-business on demand™  
ibm.com®  
iSeries™  
i5/OS™  
z/OS®  
zSeries®  
AIX®  
ClearCase®

Cloudscape™  
Domino®  
DB2 Connect™  
DB2 Universal Database™  
DB2®  
ETE™  
Informix®  
IBM®  
Lotus®  
OS/390®  
OS/400®  
Power PC®

Rational Rose®  
Rational Unified Process®  
Rational®  
Redbooks™  
RequisitePro®  
Roma®  
RUP®  
S/390®  
SoDA®  
SAA®  
Tivoli®  
WebSphere®

The following terms are trademarks of other companies:

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

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

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

# Preface

This IBM Redbook will help you plan and execute the migration of J2EE 1.3 applications developed for BEA WebLogic Server 8.1, JBoss 3.2.7 and Apache Tomcat 5.5.9, so that they will run on WebSphere Application Server V6.

This redbook provides detailed information to help you plan migrations, best practices for developing portable applications and migration working examples for each of the platforms from which we migrated.

It is not our intention to provide a feature-by-feature comparison of BEA WebLogic Server 8.1, JBoss 3.2.7 and Apache Tomcat 5.5.9 versus WebSphere Application Server V6, or to argue the relative merits of the products, but to produce practical technical advice for developers who have to migrate applications from these vendors to WebSphere Application Server V6.

## The team that wrote this redbook

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



*The team who wrote this book. From left to right, Roman Kharkovski, Leonardo Fernandez, Christian Hellsten, Hernan Cunico*

**Hernan Cunico** is a Certified Consulting IT Specialist and WebSphere® software specialist at the ITSO, Raleigh Center. He writes extensively and teaches IBM classes on WebSphere software. Hernan has 12 years of experience in the Information Technology and e-business consulting areas. His areas of expertise also include networking, security, e-business and e-commerce solutions architecture design and implementation.

**Leonardo Fernandez** is a WebSphere specialist working for IBM Global Services in Argentina. He has six years of experience in the application development field, working for the health and bank industries. He also has two years of experience delivering the implementation of Content Management solutions. His areas of expertise include application architecture, design and development.

**Christian Hellsten** is an IT Specialist with IBM Business Consulting Services. He has more than six years of experience in the IT industry working on large scale e-business projects. His areas of expertise include Java™, J2EE and EAI architecture, design and development.

**Roman Kharkovski** is a Senior Certified Technical Sales Specialist with IBM US. He has more than 12 years of experience in IT application design and development and has been working with middleware for over nine years. He holds a Master's degree in Computer Science from Moscow Engineering Physics University (MEPhI). His areas of expertise include OO design and development, J2EE servers from IBM, BEA, JBoss, Oracle and Apache, BEA TUXEDO transaction processing monitor, OLTP and Web application design and implementation, Web services, UNIX®, relational databases, Java, Pascal, C and other technologies. Roman has been a member of the Worldwide WebSphere Technical Sales Support team since 1999.

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

Wayne Beaton  
Software Services for WebSphere - IBM Canada

Andrew Blank  
WebSphere Software Lab Services - Prolifics

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or clients.

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

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

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an e-mail to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

- Mail your comments to:

IBM® Corporation, International Technical Support Organization  
Dept. HZ8 Building 662  
P.O. Box 12195  
Research Triangle Park, NC 27709-2195





# Introduction

This IBM Redbook is the latest in a series on migrating J2EE applications from different application servers to IBM WebSphere Application Server. The books in the series are:

- ▶ The IBM Redbook *Migrating WebLogic Applications to WebSphere Advanced Edition*, SG24-5956-00, focuses on migration applications from WebLogic V5.1 to WebSphere 3.5.
- ▶ The IBM Redbook *Migrating WebLogic Applications to WebSphere Advanced Edition V4*, SG24-6179-00, discusses WebLogic V6.1 and WebSphere V4.0.
- ▶ The IBM Redpaper *Migrating WebLogic Applications to WebSphere V5*, REDP-0448-00, covers migrating applications from WebLogic V7 to WebSphere V5.0.
- ▶ This redbook expands its focus and covers migration from BEA WebLogic Server 8.1, JBoss 3.2.7 and Apache Tomcat 5.5.9 to WebSphere Application Server V6.

This chapter summarizes the contents of this redbook. It also describes the process we used to identify migration issues and write this book.

## 1.1 Our objective

The migration of an Enterprise Information System is generally an expensive, complex activity that should not be taken lightly. We do not pretend to offer final and comprehensive solutions. This redbook provides some guideposts and warnings for a real migration effort. We do this largely by working through the details of several migration examples. By describing our experience and showing what tools we used, we hope to better prepare you for tackling your own migration tasks.

This book has the general goal of being a practical guide to J2EE professionals. We identify specific migration issues and make specific recommendations regarding the following:

- ▶ Migration tools
- ▶ Migration process
- ▶ Best practices: making applications more portable and more adaptable

## 1.2 Scope of this book

This book is organized around the migration of a set of J2EE 1.3 example applications. At the time of writing, the latest version of J2EE is 1.4 and J2EE 1.5 is in the works, but the majority of applications deployed today are still J2EE 1.3 based and that is why we have decided to focus on J2EE 1.3.

- ▶ We decided to use latest available version of Apache Tomcat 5.5.9, which implements the Web container part of J2EE 1.4.
- ▶ We also used the latest available version of BEA WebLogic Server 8.1, which is J2EE 1.3 certified.
- ▶ We used JBoss 3.2.7, which implements J2EE 1.3. The latest release of the JBoss 4.0 server was in late 2004, but is not deployed as widely. That is why we decided to use its previous release.
- ▶ We migrated each application using a different approach and different development tools. In some migrations, we used Rational Application Developer V6; in other migrations, we preserved the original development and build environment of the source application. In the process, we documented:
  - The step-by-step migration process.
  - Our specific use of tools.
  - The migration issues we identified along the way.
- ▶ As a last step, we analyzed, categorized, and expanded on the migration issues we encountered.

## 1.3 Applications covered in this book

This book covers several migration examples from different platforms. We have selected several interesting open source applications we have found on <http://sourceforge.net/> (xPetstore, ivata groupware) and one application built by IBM Performance Lab (Trade).

We have tried to cover as many technologies, features and functionalities for each of those platforms and samples as possible with the time and resources we had available. Table 1-1 illustrates the J2EE technologies, features and functionalities we were able to cover. The list of open source frameworks used by our sample applications is shown in Table 1-2 on page 4.

These tables are not a complete list of technologies available for these platforms or applications and they should be used as a guide to map our migration results with your own scenario.

*Table 1-1 Technologies, features and functionalities covered in the migration examples*

Application Server	BEA WebLogic Server 8.1		JBoss 3.2.7	Apache Tomcat 5.5.9	
	Trade 3.1	xPetstore EJB 3.1.3	xPetstore EJB 3.1.3	ivata groupware 0.11.1	xPetstore Servlet 3.1.3
JSP 1.2	Yes	Yes	Yes	Yes	Yes
Servlet 2.3	Yes	Yes	Yes	Yes	Yes
EJB CMP 2.0	Yes	Yes	Yes	No	No
EJB SSB 2.0	Yes	Yes	Yes	No	No
JMS 1.0.2	Yes	Yes	Yes	No	No
Java Mail 1.2	No	Yes	Yes	No	Yes
JDBC 2.1	Yes	No	No	No	No
JSP 1.2 taglibs	No	Yes	Yes	Yes	Yes

Table 1-2 Open source frameworks used in sample applications

Framework	Description	Application
Apache Commons	Utility projects for logging, bean handling, conversion of classes, etc.	ivata groupware xPetstore EJB xPetstore Servlet
Hibernate 2.1.8	Persistence framework for Java (sometimes called Object Relational Mapping tool). Used as an alternative to JDO and CMP EJBs. EJB V3 is influenced by Hibernate.	ivata groupware xPetstore Servlet
Apache Struts 1.2.4	MVC framework for Java Web applications.	ivata groupware xPetstore EJB
XDoclet 1.2	Annotation programming tool. Allows the programmer to work with a single file, instead of multiple files for single component. Used to generate Web and EJB deployment descriptors, configuration files for Hibernate from XDoclet tags within Java files.	ivata groupware xPetstore Servlet xPetstore EJB
HSQL 1.7.3	Database and JDBC driver.	ivata groupware xPetstore EJB
MySQL 4.1	Open source relational database with JDBC driver.	xPetstore Servlet
PicoContainer 1.2	Lightweight embeddable container for components that honor Dependency Injection - generic factory that can be configured dynamically. Used to manage a project's dependencies and simplify reuse of different parts of an application for external projects.	ivata groupware
NanoContainer 1.0	Container that manages trees of PicoContainers, provides classpath management and more. Used for script initialization of PicoContainer modules using Groovy.	ivata groupware
Groovy	Dynamic scripting language for the Java 2 Platform. Used to initialize NanoContainer.	ivata groupware
Apache Maven 1.0.2	Tool that extends basic capabilities of Ant and adds automatic dependency tracking, simplifies build scripting, etc. Used as a software project management tool, provides build scripting and execution.	ivata groupware

Framework	Description	Application
Apache Ant 1.5.1	Build tool for Java.	xPetstore Servlet xPetstore EJB
OpenSymphony OSCache 1.7.5	Cache facility for Web content. Used to accelerate loading of pages into the browser to reduce wait times.	ivata groupware
Apache Velocity 1.3.1	Open source template engine used instead of JSPs.	xPetstore Servlet
OpenSymphony WebWork	Open source MVC framework similar but more light-weight than Struts. Allow for pluggable view technologies and extensible configuration.	xPetstore Servlet
OpenSymphony SiteMesh	Open source layout and page decoration framework similar to Struts Tiles.	xPetstore Servlet xPetstore EJB
JUnit and JUnitEE	These are POJO and J2EE unit test frameworks that aid in test driven development	xPetstore Servlet xPetstore EJB

All migration instructions in this book are written for Microsoft® Windows® platforms, but they are not platform-specific. The same principles apply to Linux®, Unix, iSeries™, z/OS® and any other platform where you may want to run your J2EE application.

## 1.4 What is not covered in this book

During the development of this book, we had time and resource limitations that did not allow us to cover some areas of technology, such as:

- ▶ Web services
- ▶ Security
- ▶ JCA
- ▶ BMP EJB
- ▶ JMX
- ▶ Administrative scripting
- ▶ Timer and startup services
- ▶ Clustering, server administration and configuration
- ▶ Database and Operating System migration

The main focus of the book is migration and we assume that you are familiar with J2EE concepts, the Java language, the platforms we are migrating from and the development tools. As you go through the platform-specific migration chapters,

you will find references to additional documentation as needed. A complete list of all related documentation can be found in “Related publications” on page 325.

## 1.5 How to use this book

We hope you will take the time to read this entire redbook, but depending on your time and interest, each chapter can be read separately. Each chapter’s contents should be obvious from its name.

- ▶ Chapter 2, “WebSphere overview” on page 9 discusses available WebSphere Application Server V6 packages, operating systems and databases supported. This chapter also includes an overview of the Rational Application Developer V6 development tool.
- ▶ Chapter 3, “Migration strategy and planning” on page 31 includes advice for planning migration activities, discusses four migration alternatives, gives advice as to the selection of the development and migration tools, and finally covers interoperability options if you have to run multiple application servers that need to communicate with each other.
- ▶ Chapter 4, “Installation and configuration” on page 63 is a step-by-step guide to help you install and configure IBM products used in this redbook, such as DB2 Universal Database V8.2, Rational Application Developer V6 and WebSphere Application Server V6.
- ▶ Chapter 5, “Common migration issues” on page 89 lists and analyzes the migration issues we discovered. It also discusses common pitfalls found in typical applications and suggests ways to solve them.
- ▶ Chapter 6, “Migrating from BEA WebLogic” on page 111 presents step-by-step migration examples from BEA WebLogic Server 8.1 to WebSphere Application Server V6. We have migrated two different applications: Trade and xPetstore EJB using Rational Application Developer V6.
- ▶ Chapter 7, “Migrating from JBoss” on page 175 presents detailed step-by-step migration examples of xPetstore EJB application from JBoss 3.2.7 to WebSphere Application Server V6 using two different methods: WRD and XDoclet.
- ▶ Chapter 8, “Migrating from Tomcat” on page 257 presents detailed step-by-step migration examples of ivata groupware and xPetstore Servlet applications from Apache Tomcat 5.5.9 to WebSphere Application Server V6 without using the IDE.
- ▶ Appendix A, “Development tips for portable applications” on page 295 includes advice on portability and development best practices, covers most

popular design patterns and also gives a brief overview of the Rational® Unified Process® (RUP®).

Archived

Archived



## WebSphere overview

IBM WebSphere is the leading software platform for e-business on demand™. Providing comprehensive e-business leadership, WebSphere is evolving to meet the demands of companies faced with challenging business requirements, such as the need for increasing operational efficiencies, strengthening client loyalty, and integrating disparate systems. WebSphere provides answers in today's challenging business environments.

IBM WebSphere is architected to enable you to build business-critical applications for the Web. WebSphere includes a wide range of products that help you develop and serve Web applications. They are designed to make it easier for clients to build, deploy, and manage dynamic Web sites more productively.

In this chapter, we will take quick a look at the new WebSphere Application Server V6.

For a more detailed discussion of the complete WebSphere platform, please refer to the redbook *WebSphere Product Family Overview and Architecture*, SG24-6963-02.

## 2.1 About WebSphere

WebSphere is the IBM brand of software products designed to work together to help deliver dynamic e-business quickly. WebSphere provides solutions for a positive effect on a client's business. It also provides solutions for connecting people, systems, and applications with internal and external resources. WebSphere is based on infrastructure software (middleware) designed for dynamic e-business. It delivers a proven, secure, and reliable software portfolio that can provide an excellent return on investment.

The technology that powers WebSphere products is *Java*. Over the past several years, many software vendors have collaborated on a set of server-side application programming technologies that help build Web-accessible, distributed, platform-neutral applications. These technologies are collectively branded as the Java 2 Platform Enterprise Edition (J2EE). This contrasts with the Java 2 Standard Edition (J2SE) platform, with which most clients are familiar.

J2SE supports the development of client-side applications with rich graphical user interfaces (GUIs). The J2EE platform is built on top of the J2SE platform. J2EE consists of application technologies for defining business logic and accessing enterprise resources such as databases, Enterprise Resource Planning (ERP) systems, messaging systems, e-mail servers, and so forth. The potential value of J2EE to the client is tremendous. Among the benefits of J2EE are the following:

- ▶ Promotion of an architecture-driven approach to application development helps reduce maintenance costs and allows for construction of an Information Technology (IT) infrastructure that can grow to accommodate new services.
- ▶ Application development is focused on unique business requirements and rules, rather than common application aspects, such as security and transaction support. This improves productivity and shortens development cycles.
- ▶ Industry standard technologies allow clients to choose among platforms, development tools, and middleware to power their applications.
- ▶ Embedded support for Internet and Web technologies allows for a new breed of applications that can bring services and content to a wider range of clients, suppliers, and others, without creating the need for proprietary integration.

Another exciting opportunity for IT is Web services. Quite simply, Web services allow for the definition of functions or services within an enterprise that can be accessed using industry standard protocols that most businesses already use today, such as HTTP and XML. This allows for easy integration of both intra- and inter-business applications that can lead to increased productivity, expense reduction, and quicker time to market.

## 2.2 WebSphere Application Server

WebSphere Application Server is a high-performance and scalable transaction engine for dynamic e-business applications. The Open Services Infrastructure allows companies to deploy a core operating environment that works as a reliable foundation capable of handling high volume secure transactions and Web services.

WebSphere continues the evolution to a single Web services enabled, Java 2 Enterprise Edition (J2EE) application server and development environment that addresses the essential elements needed for an on demand operating environment.

The potential value of J2EE to the client is tremendous. Among the benefits of J2EE are the following:

- ▶ A simplified architecture based on standard components, services and clients, that takes advantage of the write-once, run-anywhere Java technology.
- ▶ Services providing integration with existing systems, including Java DataBase Connectivity (JDBC); Java Message Service (JMS); Java Connector Architecture (JCA); Java Interface Definition Language (Java IDL); the JavaMail API; and Java Transaction API (JTA and JTS) for reliable business transactions.
- ▶ Application development is focused on unique business requirements and rules, rather than common application aspects, such as security and transaction support. This improves productivity and shortens development cycles.
- ▶ Scalability to meet demand, by distributing containers across multiple systems and using database connection pooling, for example.
- ▶ A better choice of application development tools and components from vendors providing standard solutions.
- ▶ A flexible security model that provides single sign-on support, integration with existing security schemes, and a unified approach to securing application components.

Because different levels of application server capabilities are required at different times as varying e-business application scenarios are pursued, WebSphere Application Server is available in multiple packaging options. Although they share a common foundation, each provides unique benefits to meet the needs of applications and the infrastructure that supports them. Therefore, at least one WebSphere Application Server product package will fulfill the requirements of any particular project and the prerequisites of the infrastructure that supports it.

As your business grows, the WebSphere Application Server family provides a migration path to higher configurations.

WebSphere Application Server is available in several different packages that are designed to meet a wide range of client requirements. Each package is designed to provide a tailored environment for a specific set of clients. The following sections describe these packages for distributed platforms in more detail.

**Operating system terminology:**

- ▶ **Distributed:** Windows, Unix: AIX®, Linux/Intel®, Linux/PPC, zLinux, Solaris, HP-UX
- ▶ **iSeries:** i5/OS™, OS/400®
- ▶ **z/OS:** OS/390®

## 2.2.1 WebSphere Application Server V6 - Express

The Express package is geared toward those who need to get started quickly with e-business. It is specifically targeted at medium-sized businesses or departments of a large corporation, and is focused on providing ease of use and ease of application development. It contains full J2EE 1.4 support but is limited to a single-server environment.

The WebSphere Application Server - Express offering distinguishes itself from the other packages in that it is bundled with an application development tool. Although there are WebSphere Studio and Rational Developer products designed to support each WebSphere Application Server package, they are normally ordered independently. WebSphere Application Server - Express includes the Rational Web Developer application development tool. It provides a development environment geared toward Web developers and includes support for most J2EE 1.4 features, with the exception of EJB and JCA development environments.

However, keep in mind that WebSphere Application Server - Express does contain full support for EJB and JCA, so you can deploy applications on them.

## 2.2.2 WebSphere Application Server V6 - Base

The WebSphere Application Server - Base package is the next level of server infrastructure in the WebSphere Application Server family. Although the WebSphere Application Server is functionally equivalent to that shipped with Express, this package differs slightly in packaging and licensing. The development tool included is a trial version of Rational Application Developer, full J2EE 1.4 compliant development tool.

## 2.2.3 WebSphere Application Server V6 - Network Deployment

WebSphere Application Server - Network Deployment represents an even higher level of server infrastructure in the WebSphere Application Server family. It extends the WebSphere Application Server base package to include clustering capabilities, Edge components, and high availability for distributed configurations. These features become more important within larger enterprises, where applications tend to service a larger client base, and more elaborate performance and availability requirements are in place.

Application servers in a cluster can reside on the same or multiple machines. A Web server plug-in installed on the Web server can distribute work among clustered application servers. In turn, Web containers running servlets and JSPs can distribute requests for EJBs among EJB containers in a cluster.

The addition of Edge components provide high performance and high availability features. For example:

- ▶ The Caching Proxy intercepts data requests from a client, retrieves the requested information from the application servers, and delivers that content back to the client. It stores cachable content in a local cache before delivering it to the client. Subsequent requests for the same content are served from the local cache, which is much faster and reduces the network and application server load.
- ▶ The Load Balancer provides horizontal scalability by dispatching HTTP requests among several, identically configured Web server or application server nodes.

Table 2-1 on page 13 shows the features included with each WebSphere Application Server packaging option.

Table 2-1 WebSphere Application Server packaging and license terms

	<b>WebSphere Application Server V6 - Express</b>	<b>WebSphere Application Server V6 - Base</b>	<b>WebSphere Application Server V6 - Network Deployment</b>
Licensing terms	Limited to a max of 2 CPUS	Unlimited CPUs	Unlimited CPUs
WebSphere Application Server	Yes	Yes	Yes
Network Deployment	No	No	Yes
Not all features are available on all platforms. See the System Requirements Web page for each WebSphere Application Server package for more information.			

	<b>WebSphere Application Server V6 - Express</b>	<b>WebSphere Application Server V6 - Base</b>	<b>WebSphere Application Server V6 - Network Deployment</b>
IBM HTTP Server V6 Web server plug-ins	Yes	Yes	Yes
IBM HTTP Server	Yes	Yes	Yes
Application Client (not on zLinux)	Yes	Yes	Yes
Application Server Toolkit	Yes	Yes	Yes
DataDirect Technologies JDBC Drivers for WebSphere Application Server	Yes	Yes	Yes
Development tools	Rational Web Developer (single use license)	Rational Application Developer Trial	Rational Application Developer Trial
Database	IBM DB2® Universal Database™ Express V8.2	IBM DB2 Universal Database Express V8.2 (development use only)	IBM DB2 UDB Enterprise Server Edition V8.2 for WebSphere Application Server Network Deployment
Production ready applications	IBM Business Solutions	No	No
Tivoli® Directory Server for WebSphere Application Server (LDAP server)	No	No	Yes
Tivoli Access Manager Servers for WebSphere Application Server	No	No	Yes
Edge Components	No	No	Yes
Not all features are available on all platforms. See the System Requirements Web page for each WebSphere Application Server package for more information.			

## 2.2.4 WebSphere Extended Deployment

WebSphere Extended Deployment is built on a virtualized infrastructure that extends the traditional concepts of J2EE resources and applications, as well as their relationships with one another. This new infrastructure facilitates the ability of WebSphere Extended Deployment to automate operations in an optimal and repeatable fashion. With its automation capabilities, WebSphere Extended Deployment can help you reduce your total cost of ownership (TCO) and provide a more stable, predictable and reliable operating environment.

By extending the capabilities of WebSphere Application Server Network Deployment, WebSphere Extended Deployment can help you optimize the utilization and management of your deployments and enhance the quality of service of your business-critical applications. Major elements of the WebSphere Extended Deployment product are discussed below.

### Dynamic operations

Dynamic operations allow your application environment to scale as needed with the virtualization of WebSphere resources and the use of a goals-directed infrastructure, helping you increase the speed at which your company can adapt to business demands.

Dynamic operations include several aspects of the request flow control and allocation of resources and applications across the grid of servers in a hardware pool. The on demand router is a component that sits in front of the WebSphere Extended Deployment implementation. It represents the entry point into a WebSphere Extended Deployment topology and controls the flow of requests into the back end. Specifically, the on demand router handles the queuing and dispatching of requests according to operational policy. In optimizing queue lengths and dispatch rates, several factors are considered, including request concurrency (per node group), operational policy, service-policy weights and load balancing. Figure 2-1 illustrates the on demand router architecture.

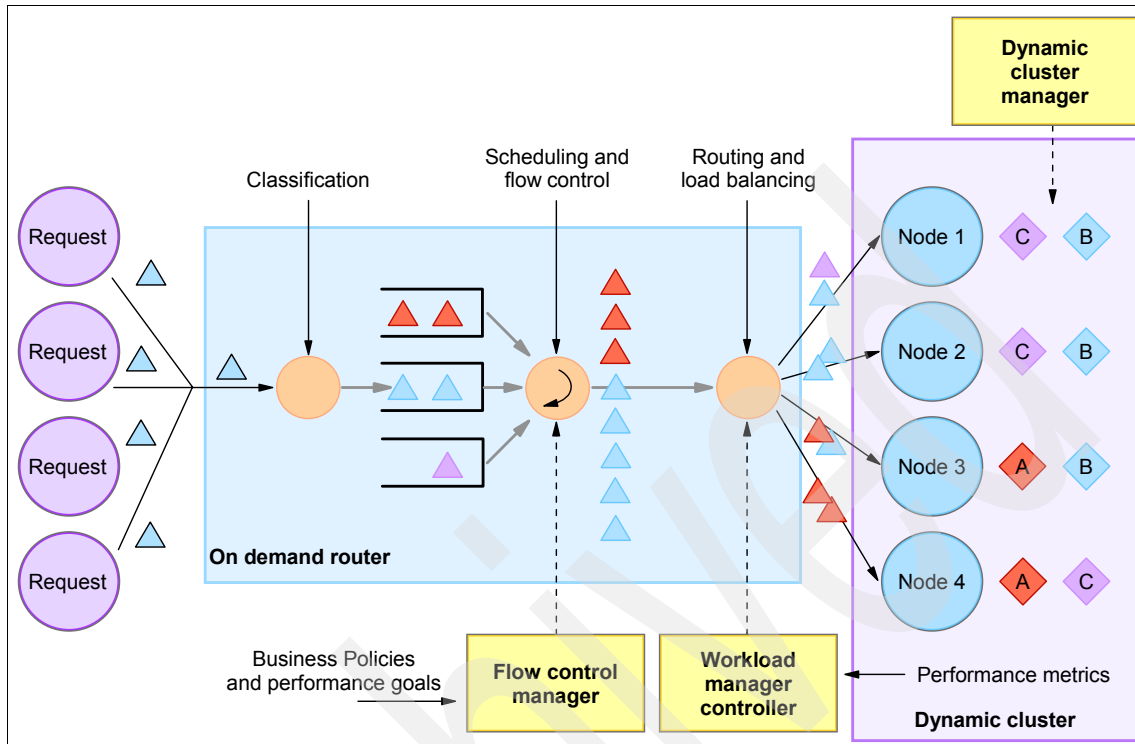


Figure 2-1 On demand router architecture

The other part of the dynamic operations capability is the resource management. Dynamic cluster managers exist in each node agent process in a WebSphere Extended Deployment topology, with only one instance being active in a node group at any given time. The primary responsibility of this autonomic manager is to control an application's footprint within a node group. Specifically, dynamic cluster managers dictate the location and cardinality of active instances in a given dynamic cluster. As demand increases or decreases, a given dynamic cluster is expanded or contracted, according to operational policy. A dynamic cluster can be deployed on any subset of the nodes in a node group except an empty set.

## High-performance computing

High-performance computing enhances the quality of service of business-critical applications to support near-linear scalability for high-end transaction processing, helping you improve client-service levels. OLTP has been, traditionally, an area where Transaction Processing monitors were the only game in town. WebSphere Extended Deployment opens this area for J2EE. One of the



techniques used to achieve scalability is partitioning facility, as is demonstrated in Figure 2-2.

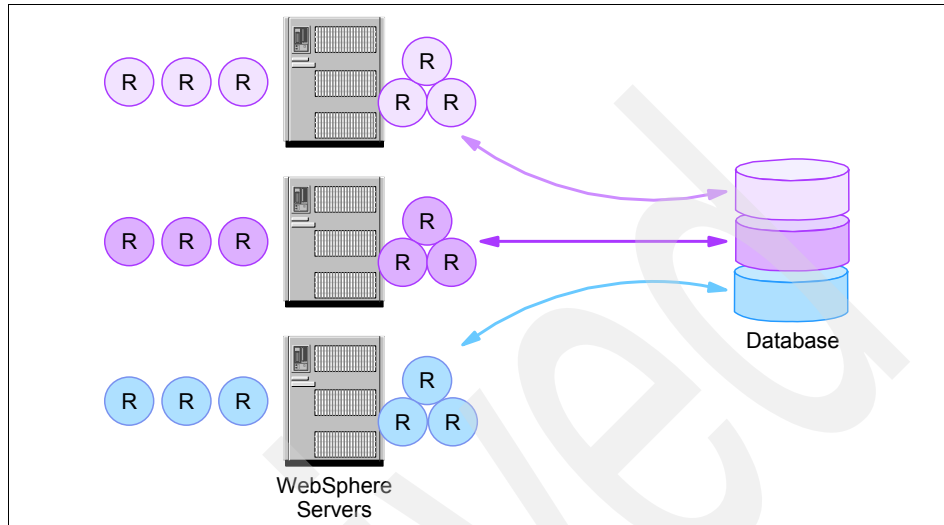


Figure 2-2 Application partitioning using the WebSphere partition facility

The partitioning pattern addresses bottlenecks that occur in high-volume online transaction processing (OLTP) applications that intensively read and write data to databases and require the most in data consistency and availability.

Examples of such systems include trading, banking, reservation and online auctioning systems. Today's J2EE servers have been optimized for read-mostly environments like commerce systems, where data-caching capabilities can be used to offload the back-end database systems. However, as systems observe increased data write (such as database insert, delete, create and update) ratios, these caching systems start to break down because of the ever-strenuous task of maintaining consistency between the cache and database. These schemes often quickly reach a point of diminishing returns and are better off sending all traffic back to the database and allowing the database to manage consistency. This strategy frequently leads to large and costly database configurations, often running on large symmetric multiprocessor (SMP) machines. In ultra-high-volume environments, these database servers inevitably become a cost and performance bottleneck.

### Extended manageability

WebSphere Extended Deployment offers simpler and improved management of complex system operations with advanced, meaningful real-time visualization tools and gradual, controlled implementation of autonomic computing capabilities, helping you reduce the cost of managing IT resources.

One of the capabilities of the extended manageability is the treemap facility. The WebSphere Extended Deployment treemap facility can be of particular interest to administrators of very large topologies, because it is well-suited to depicting large sets of data in a concise manner. For instance, the use of a set of 1000 applications could be quickly observed by viewing a treemap of dynamic clusters and comparing the sizes of rectangles. You can view goal attainment of a set of many service policies by observing the color of rectangles in a map of service policies as shown in Figure 2-3.

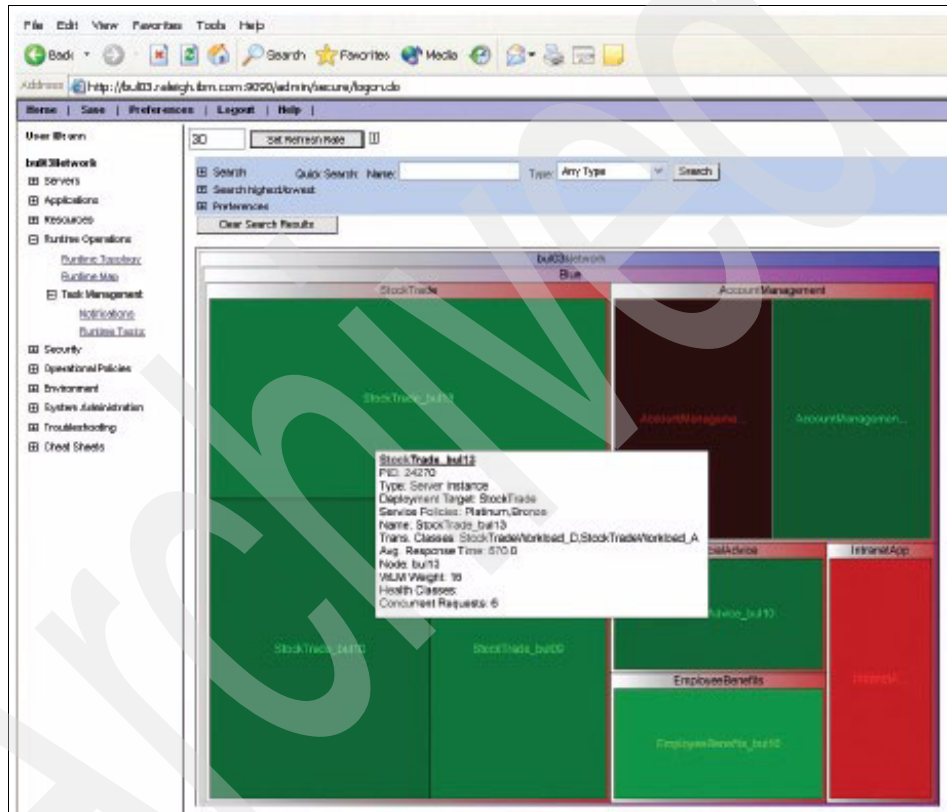


Figure 2-3 Treemap as part of administrative console

For more information about WebSphere Extended Deployment, please refer to the following URL:

<http://www.ibm.com/software/webservers/appserv/extend>

## 2.2.5 Supported platforms and software

The following tables illustrate the platforms, software and versions supported by WebSphere Application Server V6.

### Operating systems

Table 2-2 shows the supported operating systems and versions for WebSphere Application Server V6.

Table 2-2 Supported operating systems and versions

Operating Systems	Versions
<b>Windows</b>	Windows 2000 Advanced Server, Server, Professional SP4 Windows Server 2003 Datacenter, Enterprise, Standard Windows XP Professional SP1
<b>AIX</b>	AIX 5.1 Maintenance Level 5100-05 AIX 5.2 Maintenance Level 5200-02, 5200-03
<b>Sun Solaris</b>	Solaris 8 with the latest patch Cluster Solaris 9 with the latest patch Cluster
<b>HP-UX</b>	HP-UX 11iv1 with the latest Quality Pack
<b>Linux (Intel)</b>	RedHat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0
<b>Linux (Power PC®)</b>	RedHat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0
<b>zLinux</b> (only supported for WebSphere Application Server V6 - Network Deployment)	RedHat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0
<b>i5/OS and OS/400</b>	OS/400 5.3, 5.2
<b>z/OS</b> (only supported for WebSphere Application Server V6 - Network Deployment)	z/OS 1.6, 1.5, 1.4 z/OS.e 1.6, 1.5, 1.4

### Web servers

Table 2-3 on page 20 shows the supported Web servers by platform for WebSphere Application Server V6.

Table 2-3 Supported Web servers by platforms

Web servers	Platforms
IBM HTTP Server V6 (powered by Apache 2.0.x)	Windows, AIX, HP-UX, Solaris, Linux (Intel), Linux (Power PC), zLinux, OS/400, z/OS
IBM HTTP Server V2.0.43 for iSeries (powered by Apache)	OS/400
IBM HTTP Server for z/OS 5.0.1	z/OS
Apache Server 2.0.48	Windows, AIX, HP-UX, Solaris, Linux (Intel), Linux (Power PC), zLinux
Microsoft Internet Information Services 5.0, 6.0	Windows
Lotus® Domino® Enterprise Server (as HTTP server) 6.0.3, 6.5.1	Windows, AIX, HP-UX, Solaris, Linux (Intel), Linux (Power PC), zLinux, OS/400, z/OS
Sun ONE Web Server Enterprise Edition 6.0 SP7	Solaris, Windows
Sun Java System Web Server 6.1 SP1	Solaris, Windows
Covalent Enterprise Ready Server 2.4 Apache 2.0.48 Edition	Windows, HP-UX, Linux (Intel) Solaris, z/OS

## Database servers

Table 2-4 shows the supported operating systems and versions for WebSphere Application Server V6.

Table 2-4 Supported database servers and versions

Databases	Versions
IBM DB2	IBM DB2 Enterprise Server Edition 8.2, 8.1 with FP5 IBM DB2 Workgroup Server Edition 8.2, 8.1 with FP4a, FP5 IBM DB2 Information Integrator 8.2, 8.1 with FP5 IBM DB2 Connect™ 8.2, 8.1 with FP5 IBM DB2 Express 8.2, 8.1 with FP5 IBM DB2 for zSeries® 8, 7 IBM DB2 for iSeries 5.3, 5.3
Cloudscape™	Cloudscape 5.1.6x
Oracle	Oracle Standard/Enterprise Edition 10g Release 1 - 10.1.0.2 Oracle 9i Standard/Enterprise Edition Release 2 - 9.2.0.4 Oracle 8i Standard/Enterprise Edition Release 3 - 8.1.7.4

Databases	Versions
Sybase	Sybase Adaptive Server Enterprise 12.5.1, 12.0.0.8
Microsoft SQL Server	SQL Server Enterprise 2000 SP 3a
Informix®	Informix Dynamic Server 9.4, 9.3

## Directory servers

Table 2-5 shows the supported directory servers and versions for WebSphere Application Server V6.

*Table 2-5 Supported directory servers and versions*

Directory Server	Versions
IBM Directory Server	5.1 5.2
z/OS Security Server	1.6 1.5 1.4
Lotus Domino Enterprise Server (acting as LDAP server)	6.5.1 6.0.3
Sun ONE Directory Server	5.2 5.1 SP3
Windows Active Directory	2003 2000
NDS eDirectory	8.7.3

For the most updated operating system levels and requirements, refer to the following URLs:

- ▶ WebSphere Application Server - Express  
<http://www.ibm.com/software/webservers/appserv/express/requirements/>
- ▶ WebSphere Application Server  
<http://www.ibm.com/software/webservers/appserv/was/requirements/>
- ▶ WebSphere Application Server Network Deployment  
<http://www.ibm.com/software/webservers/appserv/was/network/requirements/>

## 2.3 Development and deployment tools

The WebSphere Application Server V6 environment comes with a rich set of development tools. WebSphere Application Server V6 - Express comes with the Application Server Toolkit and Rational Web Developer V6. WebSphere Application Server V6 - Network Deployment comes with the Application Server Toolkit and a trial version of Rational Application Developer V6. For a full version of Rational Application Developer V6, additional licensing is required.

This section covers the development and deployment tools that are available as well as their key features.

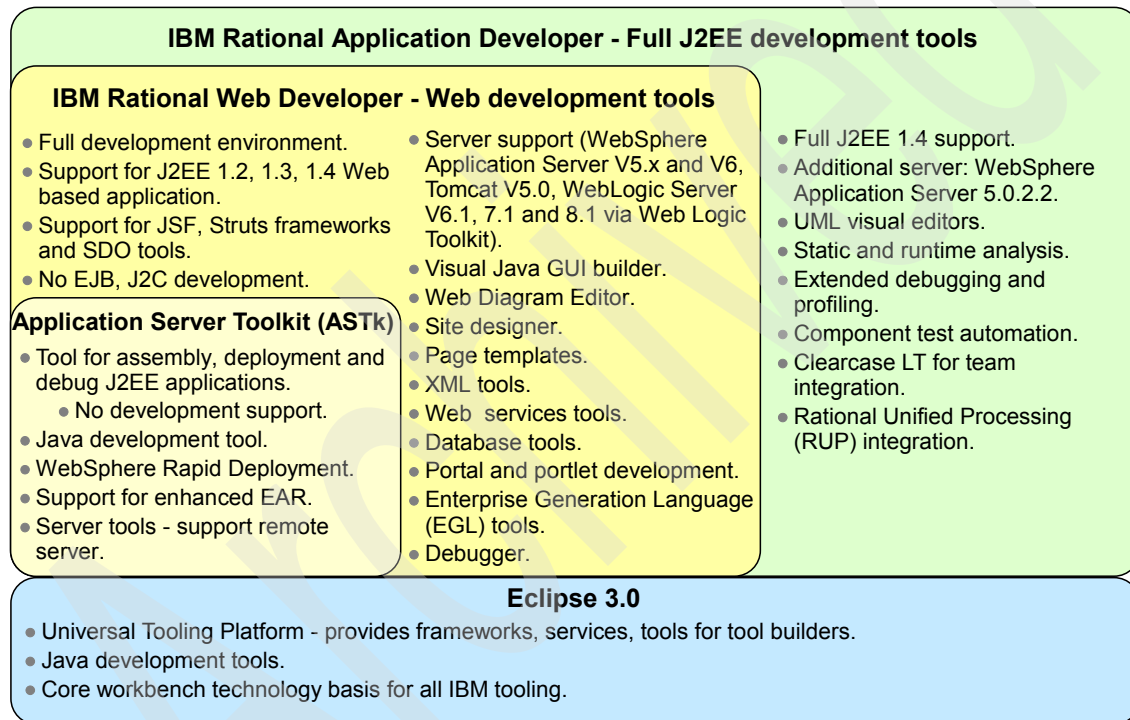


Figure 2-4 A summary of the features in Rational Web/Application Developer 6

### 2.3.1 Rational Application Developer V6

Rational Application Developer V6 includes all the subset features of Rational Web Developer V6. It is the IDE of choice for developing and deploying applications for WebSphere Application Server V6. It is the successor to WebSphere Studio Application Developer. It supports applications developed for WebSphere Application Server versions 4.0, 5.0 and 6.0.

Some of the key features of Rational Application Developer V6 are covered in the following sections.

### **Full support provided for Java 2 Enterprise Edition 1.4**

All wizards have J2EE 1.4 options. Deployment descriptor editors have been upgraded for J2EE 1.4. options. There are wizards for the migration of J2EE 1.2 and 1.3 applications to J2EE1.4 and full support of the latest enterprise Java APIs: Servlet 2.4, JSP 2.0 and EJB 2.1.

### **Annotation-based programming**

New EJB, Servlet and Web services annotations allow the programmer to use XDoclet style tags in source code files to specify configuration parameters for J2EE applications. This reduces the artifacts that the programmer has to concentrate on and keep track of.

### **Modelling functionality**

The IDE has Unified Modeling Language (UML) modeling functionality which allows the developer to express classes, EJBs and database schemas visually. New UML Topic and sequence diagrams can be used to view class relationships and method interactions. Since the diagrams are created by wizards and by dropping classes on diagram panes, only a passive knowledge of UML is required on the part of the developer. That is, the developer only needs to be able to read the diagrams rather than actively having to know the language to generate them.

### **Integration with the Rational Unified Process and Tool Set**

IBM Rational software also offers an extensive suite of tools for the end-to-end application development life cycle. These tools are not included with Rational Application Developer V6, but instead they are additional tools that can interact with the IDE.

For example, the Rational Tools Suite includes Rational ClearCase, which is a code repository solution, and Rational RequisitePro®, which is a requirements analysis tool. There are many others. Rational Application Developer V6 integrates seamlessly with these additional tools and also with the Rational Unified Process which is the iterative software development process that Rational software supports. The IDE has a Process Browser and Process Advisor view which allows the developer to view Rational Unified Process best practice advice for the task at hand.

## Application analysis

The IDE also has built-in wizards for analyzing applications for coding practices. Bad coding practices are identified in applications and examples are provided for best practices and how to resolve problems as well.

## Unit testing

Rational Application Developer V6 provides features for the easy creation, execution and maintenance of unit tests for J2EE components. Support of creating unit tests for:

- ▶ Java classes.
- ▶ EJBs (1.1, 2.0 and 2.1).
- ▶ Web services (J2EE and .Net based).

A selection of test patterns is available for defining complex unit test cases. Test data can be stored in a separate data pool for the flexible definition of test cases.

The unit test frame work in Rational Application Developer V6 is constructed using JUnit and Hyades technology. JUnit is a frame work for unit testing in Java, it is available as a separate command line tool and it is also as an Eclipse plug-in. Hyades is an Eclipse sub-project to produce an integrated test, trace and monitoring environment. Recently, Hyades has been incorporated into a larger Eclipse project for test and performance tools. This is an example of how Rational Application Developer V6 via the Eclipse IDE frame work is leveraging wider open standards technologies.

For further information, visit the JUnit and Hyades sites at the following URLs, respectively:

<http://www.junit.org>

<http://www.eclipse.org/hyades>

## Integration with Clearcase

Rational Application Developer V6 has a client for accessing Rational ClearCase LT. Rational ClearCase LT is the junior member of the Rational ClearCase family of Source Code Management (SCM) repositories. Rational Application Developer V6 also has a client for accessing CVS (Concurrent Versions System), which is another popular SCM. The Rational ClearCase LT client is another example of how the IDE integrates with the wider suite of Rational tools and the Rational Unified Process.

### 2.3.2 Rational Web Developer V6

Rational Web Developer V6 is the successor to WebSphere Studio Web Developer. It is a subset of the functionality in Rational Application Developer V6.



This sub-set of functionalities focuses on Web development tooling. Java Server Pages, Servlets, Struts, Java Server Faces, static HTML, XML and Web services development are all supported. However, if development of full J2EE applications including EJB development is required then Rational Application Developer V6 would be needed.

Some of the key features of Rational Web Developer V6 are described next.

### **Latest Web Services support**

The Web IDE allows you to build and consume Web services. Support is included for Java Bean and EJB Web services as per the Java Specification Requests JSR-101 and JSR-109. There is also support for setting a Web service conformance level to basic profile 1.0, simple SOAP basic profile 1.0 and attachments profile 1.0.

Wizards are available for generating RMI stubs and skeletons for JAX-RPC handlers.

A SOAP Monitor is integrated into the Web service creation wizard to allow the viewing of traffic in Web services.

Secure Web service support is provided by wizards to update the Web service deployment descriptors with Web Service Security settings.

### **Rapid Web development**

The Web IDE includes support for pages templates to give a consistent look and feel to Web sites. The Web site designer functionality allows the developer to nest page templates and develop navigation bars. There are also visual editors for the rapid creation of Struts applications.

### **Java Server Faces**

JSF 1.0 is now supported by the Web IDE. There are also additional IBM JSF components included and visual diagrams and editors to allow the clear layout of actions and JSF page navigation. New client components add new formats for displaying data.

### **Service Data Objects**

Service Data Objects (SDOs) offer a generic API for accessing a large number of datasources. Mediator interfaces are used to access disparate datasource formats. Data is presented in JSF front ends. This allows the Web designer to focus on presentation of data without having to concern themselves with a large number of datasource access APIs. The access to the different datasources is transparent to the developer.

## Portal application development

Portals are Web sites that provide a single point of access to applications and information. Rational Web Developer V6 provides Struts and JSF support for building portal applications. Struts is a framework for providing Model View Controller architecture for a Web application while JSF is a framework for providing 'off the peg' presentation components in a Web application. The IDE comes with Service Data Objects for accessing and updating Siebel and SAP records which can be added to portal applications.

## Enterprise Generation Language support

Enterprise Generation Language (EGL) is a high-level language for developing business logic. This language is implementation-independent and can be used by developers that are unfamiliar with specific languages such as Java and COBOL. Implementation specific code in Java or COBOL is then generated from the EGL. Rational Web Developer V6 has wizards and tools that support EGL. For more information about EGL, refer to the following URL:

<http://www.ibm.com/developerworks/websphere/zones/studio/egldocs.html>

### 2.3.3 Application Server Toolkit (ASTk)

The Application Server Toolkit is a suite of tools for deploying applications to Application Servers. The ASTk is a sub-set of functionality in Rational Web Developer V6 functionality (and hence also a sub-set of Rational Application Developer V6 functionality).

Features in the Application Server Toolkit include tools for the following operations of J2EE applications:

- ▶ Assembling
- ▶ Deploying
- ▶ Running
- ▶ Debugging
- ▶ Profiling

Applications can be assembled and then deployed to local or remote test servers. Once deployed, the applications can simply be run; they can also be debugged to diagnose errors or they can be profiled to analyze performance.

### 2.3.4 WebSphere Rapid Deployment

One of the new features in WebSphere Application Server V6 is the WebSphere Rapid Deployment. This is a feature that allows for the deployment of applications with the minimum of attention and effort on the part of the developer

and or administrator. The Rapid Deployment model has three basic features which allow for this ease of deployment:

- ▶ Annotation-based programming.
- ▶ Deployment Automation.
- ▶ Enhanced EAR.

Annotation-based programming allows the developer to annotated his EJB, Servlet or Web service module code with special JavaDoc syntax annotations (similar to XDoclet). When the source of the module is saved, the directives in these annotations are parsed and the IDE uses the directives to update deployment descriptors. This allows the developer to concentrate on the Java source code rather than meta-data files.

Deployment Automation is where applications install packages are dropped into a hot directory under an application server and the application is automatically installed. Any installation parameters not been specified by the install package's deployment descriptors have default values applied by the automated deployment process.

Rapid deployment is still more sophisticated and easy to use because it allows for a free-form deployment operation. In this mode, it is possible to drop deployment items into the hot directory without strict J2EE packaging such as Web or Enterprise archives and deployment descriptors. It is possible, for example, for annotations in the Java source code to be processed at deployment time; this information is then used to build deployment descriptors.

Finally, the rapid deployment model also includes an Enhanced EAR. This means that the Enterprise archive package can include information about resources and properties such as datasources. This allows for more information to be specified about the application and how it should be deployed, which means there are fewer decisions to be made at deployment time.

## 2.4 Performance and analysis tools

Profiling tools are available in Rational Application Developer V6 to identify and analyze various code performance problems in an application. These performance problems fall into four major categories:

- ▶ Memory leaks.
- ▶ Performance bottlenecks.
- ▶ Excessive object creation.
- ▶ System resource limits.

The information required to diagnose these problems is gathered while the application is running. The applications may be running in an Application Server or they may be simply standalone Java Applications. The application can be profiled locally or on a remote machine.

In order to profile applications on a remote machine, an Agent Controller application must be installed and running on the remote machine. The Rational IDE then connects to the Agent Controller, which then runs Profiler Agents, which in turn control and analyze the specific Java Virtual Machine in which the application is running. The Java Virtual Machine may be a JVM in WebSphere Application Server (if so, the Application Server must be started in profiling mode), or it might be a JVM running a simple standalone Java application.

Profiling sets can be configured. This means that within Rational Application Developer V6 profiling GUIs, the developer can choose which sets of profiling types are to be analyzed. For example, the developer may wish to focus on memory leak analysis rather than on code execution coverage. The Profiling sets and the profiling types that they contain allow for the fine-grained control of what type of analysis is to be run.

The subsequent sections covers the Profiling sets that are available.

### 2.4.1 Memory analysis

The memory analysis can be made of:

- ▶ Memory usage.
- ▶ Memory leak analysis manual.
- ▶ Memory leak analysis automatic.

The manual leak analysis allows more fine-grained control, whereas the automatic analysis selects default settings. The leak analysis tooling profiles the application and then saves memory heap dumps of the profile. You can select the heap dump to analyze. The tooling then identifies candidate objects that are likely candidates as the source of memory leaks. The tooling also displays the 'allocation path' for the leak candidate so that the developer can see where the memory is allocated in the path of execution of application.

### 2.4.2 Thread analysis

Thread analysis is done to examine contention of execution between threads and possible thread deadlock. In thread analysis, there are three profile sets:

- ▶ Thread view.
- ▶ UML 2 Sequence Diagram View.
- ▶ Profiling monitor.

The Thread view is a graphical representation of all threads in the application and their state. Information about locks and which threads are holding those locks is given.

The UML2 Sequence view indicates the flow of method calls. This view is synchronized with the Thread view.

The profiling monitor shows the call stack for each executing thread.

### 2.4.3 Execution time analysis

Execution time analysis is used to detect performance bottlenecks, that is, which parts of code in the execution are taking the longest to execute. Various graphical and tabular views are available to examine the execution history of the application. Representations of the entire application execution or detailed, method-level analyses are available. For example, a global execution diagram will represent the flow of execution between objects. Clicking a particular object will open a method view to show method execution within the object.

### 2.4.4 Code coverage

Code coverage is the analysis of how much each part of code in an application is executed (if at all). Code coverage can be used to identify redundant pieces of code which can be removed or refactored. The Code coverage profile set includes analysis of code execution at the method level and down to specific lines of code. The views available of this information are graphical representations of classes, methods and lines of code. Tabular and graphical representations of statistical data for code coverage are also available.

### 2.4.5 Probe kit

The probe kit is a new feature that can be used to insert Java code fragments into an application during execution. The probe kit uses the Byte-code Instrumentation (BCI) frame work to achieve this. The probes (code fragments) are saved in a separate project and can therefore be reused.

Rational Application Developer V6 provides probe editors so that probes can be easily written. When the application is profiled (executed against a profile set) the probe profile set is chosen and specific probes are chosen depending on what the profiling needs to achieve.

Probes can be inserted at the following locations:

- ▶ Method call locations.
- ▶ Method call entry and exit.

- ▶ In a catch or finally block.
- ▶ Before original code in a class static initializer.

Probes can access:

- ▶ Package class and method names.
- ▶ Method signatures.
- ▶ **this** object.
- ▶ Arguments and return values.
- ▶ Exceptions.

For example, you might write a probe that is inserted at the start and end of every method. The probe would access the class and method name and print them out to the standard output stream. This probe would then be a generic probe for tracing an application to see which parts of the code are being executed. This probe would be reusable with any application. This would save the insertion of tracing statements in an application, which can be time-consuming and misleading if an entry or exit statement (or both) is omitted by mistake. Clearly, there are many possibilities for the use of the probe kit and development teams can develop a library of probes in a separate Eclipse project, which would then be used for profiling specific Java development projects.

# Migration strategy and planning

This chapter describes the very first activities to accomplish in every migration project. This is not intended to be a final methodology, but it will help you to organize the migration process, address people about certain tasks, know where you need help or not and decide how you will go through the migration process.

This chapter also provides a list of topics that can be used as a checklist.

This chapter is organized in the following sections:

- ▶ Migration strategy considerations
- ▶ Application servers interoperability
- ▶ Migration planning activities
- ▶ Migration alternatives
- ▶ Migrating the site
- ▶ Testing the migrated site
- ▶ Going live
- ▶ Naming conventions

## 3.1 Migration strategy considerations

Migration is not just a change from one application server to another, nor does it simply mean changing some source code. Migration involves work and people from different areas and with different skills. When you migrate to a new application server, you are likely doing it because you want to take advantage of new capabilities, improve the way you manage your applications, refactor your code, improve performance or decrease downtime.

All these changes need to take place in an ordered environment, step-by-step, and with full knowledge of your target. That is why you need to plan the migration as a complete project, and like any other project, it will involve initiation, planning, execution and closeout. In this section, we discuss the initiation of the project and the activities involved.

### 3.1.1 Getting help

Depending in the complexity of your applications, your migration schedule, the skilled resources available and the critical nature of your business, you may consider getting help from IBM services team or IBM business partners.

You can outsource your complete migration project. Alternately, you can seek assistance from one or more experts, to help with migration while you provide the rest of the resources for the project. This approach has the benefit of providing mentors, who have performed a number of migrations, to help you with the actual migration tasks. The IBM WebSphere Application Server Migration Services Program offers a variety of migration services tailored specifically for WebSphere. For more details, visit:

<http://www.ibm.com/developerworks/websphere/zones/was/migration.html>

Finally, consider getting education to become familiar with the new features and how they may impact or improve your day-to-day operations.

IBM provides excellent source materials to help architects design e-business applications. For in-depth advice about application design, take a look at the Patterns for e-business site. These techniques can be helpful in a migration project:

<http://www.ibm.com/developerworks/patterns>

### 3.1.2 Migrating the different environments

No matter how many applications you have, there are three kinds of environments that you will need to migrate:



- Development environment

This group includes tools and frameworks used to develop your applications' code. Here is where the greater portion of the technical migration will be done, and this migration can be done in different ways as discussed in 3.1.3, "Strategies to handle migration complexity" on page 33. It is recommended that you have software version control and backup copies of your work as you progress through the project.

- Test environments

Every test environment is included in this group: system test, performance test, pre-production, etc. Some of them should be as close as possible to the production environment configuration in order to obtain more realistic results, especially if you are focused on performance. Generally, these test environments are developed initially for deploying and testing the application being developed, but the same environments can be used to test the application migration.

- Production environment

The production environment is where the applications are actually running. This is the most sensitive environment since migration will have a direct impact on your business. The magnitude of this impact will depend on the migration approach you decide to use. This decision will be based on several factors, such as hardware availability and business requirements, among others. Refer to 3.5, "Migrating the site" on page 58 for further details about the migration approaches.

Identify one development team and assign it the task of migrating one simple application. This task should not be complicated, but it will give the team a feel for the new platform, the differences with the previous one, as well as hands-on experience; this will also give you a measure of the time needed to migrate.

Consider that not all the development teams are the same; the applications could be very different (not only in size, but in complexity) and they may have different delivery schedules, so, in turn, you might have to plan to migrate the applications at different times, having both application servers running concurrently and interoperating for a period of time. The issues of interoperability are covered in "Vertical slice strategy" on page 35.

### 3.1.3 Strategies to handle migration complexity

Once you have the first application ready to migrate, you have to decide on a migration strategy. Based on the belief that a small problem is easier to solve than a large one, we discuss three different methods to achieve this strategy. In some cases, you will need to apply more than one method to get to the most atomic form of the problem. These migration strategies are:

- ▶ Modularization strategy
- ▶ Deploy, test and solve strategy
- ▶ Vertical slice strategy

## Modularization strategy

This is a common practice that is enforced by many technologies. J2EE divides the application into different archives inside an EAR file. These different archives are part of the problem of building the application, dividing it in business logic and business presentation. So, the first approach to migrating the application could be to migrate each of those modules separately.

Probably the less difficult kind of module to migrate will be the jar file containing the utility classes, followed by the war file with the Web application and ending with the jar file containing the EJBs.

However, the migration path suggested will not be in order of complexity, but in order of testability. Here is the suggested path:

1. Utility classes  
This kind of module should be the simplest to migrate, since it usually does not use many features of the Application Server, but simply provides utility issues classes or solves data conversion problems (think of it as Apache Commons). You can test the classes included in those jar files using JUnit, which is supported by Rational Application Developer V6.
2. Enterprise Java Beans  
Once you have all the utility classes you use inside your EJBs migrated, it is time to migrate the EJBs. EJB migration is perhaps the most difficult part of the migration, because EJBs use many features from the Application Server, and the J2EE specification leaves some definitions open to vendor interpretation. That is why there are different XML files to configure the same element across the different J2EE compliant servers.  
  
Once you have your EJBs migrated, you can deploy the jar file and test your EJBs with the Universal Test Client or using JUnit; these are both provided with Rational Application Developer V6. You can also use other testing frameworks such as JUnitEE or Cactus.
3. Application Client  
If there is any Application Client Project to migrate, it should be migrated after the EJBs.
4. Web application  
At this point, you have utility classes and EJBs migrated and tested, so the only thing left to do is migrate the Web application.

As you perform the migration, we suggest that you write extensive tests and integrate them as part of your regular build process. The ultimate measure is

Test Driven Development, popularized by the eXtreme Programming (XP). As you finish large parts of the migration, you should also perform a Functional Verification Test (FVT), and a System Integration Test (SIT).

### **Deploy, test and solve strategy**

This kind of approach can be used in simple, well coded, or almost migrated applications. It consists of deploying the application and testing it until an error occurs, going back to the application, solving the problem, and deploying the application again. This technique is rudimentary, but if you want to keep working without an IDE, or you do not have the source code of the application, this may be the best approach.

### **Vertical slice strategy**

The same concept used in the development of an application can be used in its migration. The idea is to take an important use case of the application and migrate it as a separate application. It will only contain limited functionality, but it will be migrated end-to-end. The problem with this kind of migration is that it requires a deep understanding of the application to know which slice to migrate. This should be the less cohesive one, to ensure the fewest code modifications possible to simulate the other non-migrated parts of the application.

Very often, the application that needs to be migrated is very complex and cannot be migrated all at once. This is why you slice it into parts and perform a phased migration, potentially over multiple months for very large projects. Over a period of time, different parts of application may need to communicate with each other. For example, in an online commerce application, you may migrate the shipping module first and make it available to the checkout service that may still be running in the original environment. In some cases, you might need bi-directional connectivity between the two. You might have a need for security, transactions, synchronous and asynchronous communications. There are also different requirements for performance and footprints in terms of the software licenses and hardware utilized. These requirements may vary widely and this is why it is difficult to make generic, one-size-fits-all recommendation as to the interoperability option to be used.

In some cases, a complete migration may take several months; in other cases, parts of applications will never be migrated. There are also clients who decide to perform all new development and deployment with IBM software, but keep their existing applications intact. It is likely that these existing applications need to communicate with the new ones, as in the case of a phased migration. This leads us into the interoperability discussion.

## 3.2 Application servers interoperability

In recent years, the concept of universal interoperability became widely accepted. It is called Enterprise Service Bus (ESB). IBM and many other vendors see ESB as an architectural pattern that can be implemented in different ways using different protocols and products.

In Table 3-1 on page 38, Table 3-2 on page 40 and Table 3-3 on page 41, we have tried to summarize some of these considerations as they apply to J2EE application servers. Following is a description of each protocol that may be used to interoperate between different application server instances as well as between different vendor products.

### ► BEA T3

This is a proprietary BEA protocol similar to Java RMI in API, but different on the wire and which does not interoperate with RMI. It is used extensively by internals of WebLogic and is available for WebLogic client applications. Many clients using this in production invoke WebLogic EJBs from their WebSphere servers. All you need to do is to put the WebLogic T3 provider jar file into your application classpath and use BEA WebLogic INITIAL\_CONTEXT\_FACTORY and PROVIDER\_URL. For more details, see the following URL:

[http://e-docs.bea.com/wls/docs90/rmi/rmi\\_t3.html](http://e-docs.bea.com/wls/docs90/rmi/rmi_t3.html)

### ► JMS over WSMQ

The WSMQ product provides JMS as one of its programming APIs. WSMQ can be used as a JMS provider from within any J2EE application server, C, COBOL, and supports over 40 platforms. WSMQ provides great QoS for its messages and is one of the fastest messaging engines with 80% market share in the MOM market. It should be noted that the performance of this protocol is heavily dependent on the quality of services configured in the messaging engine. Because of the persistence and assured delivery options, we did not give it the highest mark for performance. This is only because we compared it to low latency synchronous protocols that do not have high quality of service. If you need very high performance and do not require full crash recovery, you can configure non-durable in-memory messaging and achieve very high performance levels comparable to those of synchronous protocols. This is a very popular choice for interoperability between WebSphere and WebLogic as well as other application servers. For more details, see the following URL:

<http://www.ibm.com/software/integration/wmq>

### ► SOAP

Simple Object Access Protocol (SOAP) is a popular Web services communication protocol. most often used over HTTP, although it can also use

JMS, SMTP or other lower level protocols. Recent developments in Web services have added a great deal to its capabilities, so it hardly deserves the name “Simple” anymore. These capabilities include security, transactions, policies, reliability, registry and publish/subscribe services. SOAP is becoming more and more popular. WS-I.org is a good resource to learn more about interoperability options for SOAP. Many clients use this protocol in production to connect multiple heterogenous systems together.

- ▶ RMI-IIOP (EJB only)

This protocol is part of the J2EE specification 1.2 and above. It is designed for EJB clients to allow them to invoke EJBs over a CORBA IIOP stack with automatic security and transaction flow, subject to the implementation and support terms of the vendor. This is an efficient synchronous protocol, but unlike SOAP-HTTP, it is not well supported by firewalls. RMI-IIOP is very popular to connect multiple domains of the same server and less popular for heterogenous interoperability.

- ▶ IIOP

This is a pure CORBA IIOP protocol as defined by OMG. There are very few applications built as CORBA and WebSphere allows non-Java EJB clients to invoke EJBs via IIOP. This is not very popular protocol and you probably need to consider other alternatives.

- ▶ 3rd party JMS

This option assumes you have a third-party JMS provider, such as ActiveMQ, Sonic, Tibco or any other JMS server. You will have to configure the JMS provider in both application server environments, making sure you have transactional JTA support for sending/receiving messages and that it works under a high load. This option is not used very often because most organizations who use messaging tend to prefer WSMQ.

- ▶ HTTP socket

This is an “ancient” way of connecting to the server by custom programming and opening direct connection from the client, using HTTP to access the target system in a similar way to a browser, except that instead of HTML payload, some custom protocol and encoding is being used. This approach is virtually extinct these days since new applications are built much more easily with standardized SOAP and WSDL that have great tool support and are very extensible. We do not recommend this option to anyone except highly specialized systems.

We have used a very simple binary scheme to rate all protocols against each other in separate QoS categories. Our ratings are simple: Y and N.

- ▶ “Y” means a better choice as it relates to other protocol choices in the table.

- “N” designates a lower ranking, but not necessary a resounding *no*. It simply means that the rated protocol offers a somewhat lesser capability in a particular category, but it still may be a viable choice when considered across all QoS categories.

We have put quality of service descriptions into the right-hand side of our tables. A detailed description of QoS used in these tables is given at the end of Table 3-3 on page 41.

Table 3-1 Interoperability between BEA WebLogic Server 8.1 and WebSphere Application Server V6

Protocol	Description	Synchronous	Bidirectional	Strategic	Supported	Tested	Best performance	Easy to set up	Low cost	Loose coupling	Security propagation	JTA (2PC) support	Client transaction
BEA T3	<p><u>WebSphere Application Server as client for WebLogic Server:</u> need to code Java client for BEA T3 (just like RMI). Any version of WebSphere Application Server can be used as a client for any version of BEA WebLogic Server. You simply need to put proper WebLogic Server T3 jar files on the WebSphere Application Server classpath. No changes needed on the WebLogic Server side.</p> <p><u>WebSphere Application Server as server for WebLogic Server:</u> T3 is BEA proprietary protocol not supported by WebSphere Application Server.</p>	Y	N	N	N	Y	Y	Y	Y	N	N	N	Y
JMS over WSMQ	<p><u>WebSphere Application Server as client for WebLogic Server:</u> need to code Java JMS client to send/receive messages. For WebLogic Server, need to setup MQ bridge and code MDB or pull messages via JMS Java calls.</p> <p><u>WebSphere Application Server as server for WebLogic Server:</u> need MDB or JMS calls from Java. On WebLogic Server, need to set up MQ bridge and code JMS Java client to send/receive messages.</p>	N	Y	Y	Y	Y	N	N	N	Y	N	N	N
SOAP	<p><u>WebSphere Application Server as client for WebLogic Server:</u> need to import WSDL and generate Web services client, subject to version limitations and Java types used (see WS-I.org profiles). WebLogic Server does not need to be aware of the client and does not need any changes.</p> <p><u>WebSphere Application Server as server for WebLogic Server:</u> same as above, but in reverse.</p>	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N	N

Protocol	Description	Synchronous	Bidirectional	Strategic	Supported	Tested	Best performance	Easy to set up	Low cost	Loose coupling	Security propagation	JTA (2PC) support	Client transaction
RMI-IIOP (EJB only)	<u>WebSphere Application Server as client for WebLogic Server:</u> no change in client Java source code is needed. Can invoke WebLogic EJBs from WebSphere Application Server V5 and above, but need proper initialContext. WebLogic EJBs do not need to be changed and are not aware that they are being called by a WebSphere Application Server client.	Y	Y	Y	N	Y	Y	Y	Y	N	Y	N	Y
	<u>WebSphere Application Server as server for WebLogic Server:</u> same as above, but in reverse.												
IIOP	<u>WebSphere Application Server as client for WebLogic Server:</u> need to code old fashion way, CORBA specific Java client. Server is not aware of the client specifics, but formal BEA support may not be available.	Y	Y	N	N	Y	Y	Y	Y	N	N	N	Y
	<u>WebSphere Application Server as server for WebLogic Server:</u> WebSphere Application Server supports party ORB clients for EJB calls without changes to EJBs. WebLogic client will have to code old fashion way Java CORBA client.												
3 <sup>rd</sup> party JMS	<u>WebSphere Application Server as client for WebLogic Server:</u> similar to JMS over WSMQ option.	N	Y	N	N	N	N	N	N	Y	N	N	N
	<u>WebSphere Application Server as server for WebLogic Server:</u> same as above.												
HTTP socket	<u>WebSphere Application Server as client for WebLogic Server:</u> in Java client need to manually open socket connection and send HTTP request to custom WebLogic remote URL acting as a reverse proxy for services on WebLogic. This was usually done before SOAP became popular and is not recommended. Need to develop servlet acting as proxy and map incoming HTTP requests to EJBs, invoke EJB and generate HTTP responses back to the client.	Y	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
	<u>WebSphere Application Server as server for WebLogic Server:</u> same as above, but in reverse												

Table 3-2 Interoperability between JBoss 3.2.7 and WebSphere Application Server V6

Protocol	Description	Synchronous	Bidirectional	Strategic	Supported	Tested	Best performance	Easy to set up	Low cost	Loose coupling	Security propagation	JTA (2PC) support	Client transaction
JMS over WSMQ	<p><u>WebSphere Application Server as client for JBoss:</u> need to code Java JMS client to send/receive messages. For JBoss need to set up the MQ connection factory and code MDB or pull messages via JMS Java calls.</p> <p><u>WebSphere Application Server as server for JBoss:</u> need MDB or JMS calls from Java. On JBoss, need to setup MQ as JMS provider and code JMS Java client to send/receive messages.</p>	N	Y	Y	Y	N	N	N	N	Y	N	N	N
SOAP	<p><u>WebSphere Application Server as client for JBoss:</u> need to import WSDL and generate Web services client - subject to version limitations and Java types used (see WS-I.org profiles). JBoss server (uses AXIS as SOAP provider), does not need to be aware of the client and does not need any changes.</p> <p><u>WebSphere Application Server as server for JBoss:</u> same as above, but in reverse.</p>	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N	N
RMI-IIOP (EJB only)	<p><u>WebSphere Application Server as client for JBoss:</u> no change in client Java source code. Should be able to invoke JBoss EJBs from WebSphere Application Server V5 and above, but need proper initialContext. JBoss EJBs do not need to be changed and are not aware of the WebSphere Application Server client.</p> <p><u>WebSphere Application Server as server for JBoss:</u> same as above, but in reverse.</p>	Y	Y	Y	N	N	Y	Y	Y	N	Y	N	Y
3 <sup>rd</sup> party JMS	<p><u>WebSphere Application Server as client for JBoss:</u> similar to JMS over WSMQ option.</p> <p><u>WebSphere Application Server as server for JBoss:</u> same as above.</p>	N	Y	N	N	N	N	N	N	Y	N	N	N



Protocol	Description	Synchronous	Bidirectional	Strategic	Supported	Tested	Best performance	Easy to set up	Low cost	Loose coupling	Security propagation	JTA (2PC) support	Client transaction
HTTP socket	<p><u>WebSphere Application Server as client for JBoss</u>: in the Java client need to manually open a socket connection and send HTTP request to custom JBoss remote URL, acting as a reverse proxy for services on JBoss. This was usually done before SOAP became popular and is not recommended. Need to develop a servlet acting as a proxy and map incoming HTTP requests to EJBs, invoke EJB and generate HTTP responses back to the client.</p> <p><u>WebSphere Application Server as server for JBoss</u>: same as above, but in reverse.</p>	Y	Y	N	Y	Y	Y	Y	Y	Y	N	N	N

Table 3-3 Interoperability between Apache Tomcat 5.5.9 and WebSphere Application Server V6

Protocol	Description	Synchronous	Bidirectional	Strategic	Supported	Tested	Best performance	Easy to set up	Low cost	Loose coupling	Security propagation	JTA (2PC) support	Client transaction
JMS over WSMQ	<p><u>WebSphere Application Server as client for Tomcat</u>: need to code a Java JMS client to send/receive messages. For Tomcat, need to setup MQ connection factory and pull messages via JMS Java calls (no MDB support).</p> <p><u>WebSphere Application Server as server for Tomcat</u>: need MDB or JMS calls from Java. On Tomcat need to set up MQ as JMS provider and code JMS Java client to send/receive messages.</p>	N	Y	Y	Y	N	N	N	N	Y	N	N	N
SOAP	<p><u>WebSphere Application Server as client for Tomcat</u>: need to import WSDL and generate Web services client - subject to version limitations and Java types used (see WS-I.org profiles). Tomcat hosts AXIS as the SOAP provider and does not need to be aware of the client.</p> <p><u>WebSphere Application Server as server for Tomcat</u>: same as above, but in reverse.</p>	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N	N

Protocol	Description	Synchronous	Bidirectional	Strategic	Supported	Tested	Best performance	Easy to set up	Low cost	Loose coupling	Security propagation	JTA (2PC) support	Client transaction
3 <sup>rd</sup> party JMS	<u>WebSphere Application Server as client for Tomcat</u> : similar to JMS over WSMQ option.	N	Y	N	N	N	N	N	N	Y	N	N	N
	<u>WebSphere Application Server as server for Tomcat</u> : same as above.												
HTTP socket	<u>WebSphere Application Server as client for Tomcat</u> : in the Java client need to manually open a socket connection and send HTTP request to custom Tomcat remote URL acting as a reverse proxy for services on Tomcat. This was usually done before SOAP became popular and is not recommended. Need to develop a servlet acting as a proxy and map incoming HTTP requests to POJOs, invoke them and generate HTTP responses back to the client.  <u>WebSphere Application Server as server for Tomcat</u> : almost same as above (can map requests to EJBs or other type of service in WebSphere Application Server)	Y	Y	N	Y	Y	Y	Y	Y	Y	N	N	N

Here are QoSs we have used in the interoperability tables above:

► Synchronous

This column simply refers to the protocol ability for non-blocking calls. In the case of RMI-IIOP, all calls are synchronous, while in the case of JMS, calls are asynchronous. Please note that synchronous interaction can be built based on two asynchronous calls. Also note that in many cases, greater performance can be achieved by splitting the work into multiple parallel requests using non-blocking asynchronous calls.

► Bi-directional

This capability refers to the fact that both client and server should be able to initiate the conversation. For example, in the case of the T3 protocol, the WebSphere or JBoss or Tomcat client is able to invoke EJBs in BEA WebLogic Server 8.1, but the WebLogic client is not able to invoke WebSphere or JBoss or Tomcat since none of these supports this T3 protocol. In another example, WSMQ is supported by all of these servers and thus provides bi-directional communication channel, no matter what product acts in a server and client role.

► Strategic

This topic is introduced to give guidance for your future technology direction. If you have to decide on the protocol, we suggest that you consider the industry and vendor direction and set your timelines in line with the future support options for this protocol by vendors. Prime examples of strategic protocols are SOAP and WSMQ.

► Supported

This refers to the fact that the vendor may or may not provide formal support for the particular protocol if you have a problem. We could have split it into two columns for IBM support and “Other vendor” support, but we merged it into one by using the AND operation, so that is you will see Yes only if both vendors support the protocol.

► Tested

Yes in this column indicates that we have some direct or anecdotal evidence that this protocol for the combination of products was attempted and was proven to work at least in the lab environment. This does not refer to the formal rigorous tests that IBM or other vendors typically perform. This column is a more informal measure. Perform your own tests to verify that the protocol still works with the version of products you are using in your own particular hardware, network and software environment.

► Best performance

This column designates a *relative* measure of protocols against each other. In some environments, even N may be more than enough. In other environments, even the best performing protocol may be used inappropriately and have very poor performance. For example, if you access an EJB over the network for every getXXX() and setXXX() operation separately and do not use the Data Access Object pattern, you may get a very poor performance. The same applies to SOAP and any other network communication.

► Easy to set up

This refers to the fact that you may have to purchase, install, configure and maintain additional layers of software to provide support for some protocols. For example, with WSMQ, you will have to install and configure additional software, while for T3 or SOAP or RMI-IIOP you already have all necessary binaries as part of your application server install. Having said that, WSMQ gives you freedom and capabilities that application server protocols do not provide. It is a question of trading quality of service for simplicity and cost.

► Low cost

This includes all kinds of costs that may be associated with the choice of the particular protocol. This includes cost of development (is it easy to program for the protocol?), cost of setup and configuration, cost of additional software

required, cost of maintenance, cost of risk of lost messages and downtime. It is all relative and it is hard to give a generic measure. For example, with WSMQ, you have to factor in the cost of the license and installation and (potentially) additional hardware, so it seems expensive, but this may not be accurate for some environments where a single message carries a lot of value and you cannot afford to lose your messages. This is called downtime cost or risks cost.

- Loose coupling

This capability refers to the fact that the server does not need to know about the client and vice-versa. When the server implementation changes but the API stays the same, the client does not need to be recompiled. Also, the version of the server runtime may change, but the client does not need to know about it. For example, if you have upgraded your server from BEA WebLogic Server 5 to BEA WebLogic Server 8.1, your client running inside of WebSphere Application Server V6 that uses the WebLogic service does not need to have new jar files installed or be recompiled.

- Security propagation

This category describes the capability of the protocol of automatically flowing security context without programming. The prime example of this capability is CORBA CSiv2, which is part of J2EE 1.3 and is supported by RMI-IIOP. WebSphere Application Server V6 also supports automatic security context flow over the Web services calls with WS-Security. This is possible because WebSphere Application Server V6 supports WS-I Basic Security Profile. If the security context is not automatically propagated by the client container or is not understood by the server container, there are still options to manually inject such context into the request flow, but this may lead to an interface bloat (too many parameters on the interface). For an example of such an experiment, see this article:

<http://submit.boulder.ibm.com/dd/wsdd/asis/501650/501650.html>

- JTA (2PC) support

This refers to the capability of propagating Two Phase Commit (JTA) transaction context automatically over the stack of calls. This is a similar concept to the Security one described above, but as applied to the transactions. The difference is that there is no workaround for this if your server does not support external transactions. RMI-IIOP is capable of carrying the transaction context, but not all J2EE servers accept external transactions. Here is the quote from the WebSphere Information Center article: *“WebSphere Application Server is a transaction manager that supports transactional interoperation with other transaction managers through either the CORBA Object Transaction Service (OTS) protocol (for example, application servers) or, for JSR-109 compliant requests, Web Service Atomic Transaction (WS-AtomicTransaction) protocol. This is in*

*addition to its ability to coordinate XA resource managers and to be coordinated by J2EE Connector 1.5 resource adapters.” This quote was found at the following URL:*

[http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjta\\_intop.html](http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjta_intop.html)

In case you decide to use WSMQ as the protocol of choice, you won't be able to achieve 2PC for the end-to-end transaction, but you can make your message send and receive transactional operations and configure WSMQ for durable transactional messaging. This will provide the assured delivery of messages and split your one logical transaction into at least two physical transactions: one on the client side and one on the server side. If you have a need for a rollback of such a logical transaction, you will have to implement a compensation transaction. For example, you may have to send a rollback message that will undo the previous action.

**Tip:** WebSphere Business Integration Server Foundation V5.1 provides a compensation transactional framework for long-running processes that use Web services, EJBs, JMS, JCA and JDBC.

► Client transaction

This capability refers to the client being able to start/commit/rollback a transaction on the server, except that in this case, the server transaction is independent of the client global transaction. For example, if you have an EJB running in BEA WebLogic Server 8.1 that needs to invoke an EJB running in WebSphere Application Server V6, can this “client” EJB initiate and commit transactions on the WebSphere side? In this case, the outcome of the WebSphere transaction has no direct impact on the current transaction context of the WebLogic EJB.

### 3.3 Migration planning activities

As part of the migration planning, there is some information to gather in order to understand its complexity, discover migration issues, and understand the skills needed to accomplish it. The following activities represent the steps to gather all the information needed in the migration.

- Gathering concerned parties together
- Evaluating current assets
- Assessing the high-level application architecture
- Reviewing and validating the application code
- Reviewing the development environment
- Reviewing the runtime environment

- ▶ Reviewing the current build and deployment processes
- ▶ Assessing current skills
- ▶ Reviewing time constraints
- ▶ Creating a detailed migration plan

### 3.3.1 Gathering concerned parties together

The migration assessment is a good opportunity to bring concerned parties together. An interesting result of assessment is often a clearer understanding between the different groups about the larger picture, and a greater appreciation for the issues faced in other areas. At this time, it is a good idea to define your project goal and set expectations and measures of success, as well as a timeline and management structure. This phase is no different from that in any software project.

### 3.3.2 Evaluating current assets

Assess the current state of the applications targeted for migration. Determine which assets are involved in this migration.

#### ▶ Topology

Topology refers to what devices and computers are been used to set up the applications, the physical layout of each one and the relationship between them. Assess the current topology and decide if you want to keep it as it is. This is a good opportunity to decide to change the current topology, taking advantage of the capabilities of WebSphere Application Server V6. For further details about available topologies, refer to *WebSphere Application Server V6 Planning and Design*, SG24-6446.

#### ▶ Hardware

Assess the power of the computers involved, processors assigned, storage needed, and bandwidth used.

#### ▶ Source code

For every application to be migrated, follow this checklist:

- Is it a custom application?
- Is it being maintained?
- Is it in development process?
- Do you have the source code of the application?
- Does it use third-party modules? If so, are they supported in the target environment?

#### ▶ Business processes

Determine the business process related to every application to be migrated. Measure the allowed downtime of every application and the business areas using every one of them. If possible, let the most critical applications migrate late, once you have gained some experience migrating the less critical ones. On the other hand, it may be a good idea to migrate very simple application first to gain initial experience and then migrate the most complex part, so that the rest of the migration effort is more or less a mechanical task. This may be a very good approach if you hire expensive consultants or mentors for the beginning of your project and wish to finish the bulk of the migration on your own.

- Integration with other products

Assess the applications integrated with other products and the technology used to integrate. Determine whether that technology is standard and supported, or if there are another technologies to connect to the other product. The main issue to look for here is access to existing systems. That will be one of the most important issues of the migration.

### 3.3.3 Assessing the high-level application architecture

With the gathering of concerned parties together comes an excellent opportunity to articulate the application architecture at a high level. With a clearer view of the big picture, the stage is neatly set for subsequent parts of the assessment. A high-level architecture review should not take more than a few hours.

### 3.3.4 Reviewing and validating the application code

This is not a full code review. The idea behind this review is to find potential migration issues, not determine code quality. Of course, the higher the quality of the code, the simpler it will be to understand it and to find possible migration issues. Consider assigning time for the review of the code of the application. Depending on the complexity of the application, this could take from one to several days. If possible, include the application architect in the reviews.

#### Verifying the baseline

Although it should be obvious, we recommend that you first verify that the application works as expected on the source system: BEA WebLogic Server 8.1, JBoss 3.2.7 or Apache Tomcat 5.5.9 in the case of our book. We applied this strategy to the examples in this book and it helped us be more productive and perform migrations more quickly by identifying problems at an early stage. This helps ensure the following:

- You have collected all of the necessary artifacts.
- All the pieces actually work together.

It is a common problem during migration to deal with missing or incorrect artifacts. If the migration team does not follow this strategy, it may get into trouble. Applications that need to be migrated may have compile errors, incorrect build scripts, errors in its deployment descriptors, invalid XML files and JSP pages and many more problems. Using the incorrect artifacts wastes time and resources.

In some cases, it is advisable to have as little variation between the source and target environments as possible. For example, if your source application runs on JBoss 3.2.7 with MySql and your target environment is WebSphere Application Server V6 with DB2 Universal Database V8.2, then we suggest that you make your application work with DB2 on JBoss and then start migrating it into WebSphere.

### **Code organization**

The organization of the code might sound like a simple matter of order, but it could take several hours to organize the code to fit it in a logical structure, and put every source code in the right directory, inside a project. Assess the organization of the code for every application to be migrated, in order to determine the need to rearrange the structure. This need will be related to the migration strategy assigned to the application. Code organization is often driven by your build process, therefore you have to decide if you want to keep your existing build or if you need a new approach.

### **Changes to the specifications**

The main reason why the migration is possible is that the applications were built following a standard. The J2EE specification is evolving with time, adding new functionality and making it more flexible. But those changes have to be followed by your application in order to stay up-to-date. Gather the versions of each technology used in every application. This will give you an idea of the changes to be made to fit the new J2EE specification.

As a second step of your migration, you may want to migrate your application to J2EE 1.4 to take advantage of the new functionalities of this new specification. WebSphere Application Server V6 has completed the full J2EE certification test suite, and supports all the J2EE 1.4 APIs. Keep in mind that J2EE 1.4 is backward-compatible, so you should be able to run your J2EE 1.3 applications without change.

Pay attention to the applications that were developed for non-compliant applications servers, or using functionality that is not included in the standard. Once again, the advice is to follow the standards and make applications flexible and adaptable.



## Use of proprietary technologies

One of the most important issues of the migration is the use of proprietary technologies. The use of proprietary code will be translated in the need to code that functionality again. Pay special attention to this point. Assign time to make a proof of concept for migrating the most difficult technologies. Those proofs of concept will ease the migration of the actual application.

## Third-party libraries

Get information about the use of third-party libraries in every application. Check the version used, and whether it is supported. Again, the payback of the time assigned to proofs of concept will be significant in the actual application migration time.

### 3.3.5 Reviewing the development environment

Once you have all the code reviewed, you have to decide which migration tool you will use. There are several options available.

- ▶ Rational Application Developer V6 is perhaps the most powerful tool, since it will help you identify migration issues and solve them. It will also provide tools and wizards to create new code, or refactor the existing code, simplifying the tasks of migration and development, adding visual aid to the creation of the deployment descriptors. You can use the built-in migration plugin provided by IBM to upgrade your applications from J2EE 1.3 to J2EE 1.4. There is also a migration plugin to move applications from WebLogic to WebSphere; this is discussed in more detail in “Installing the J2EE Competitive Migrator Plugin 1.2.0 plugin” on page 125.
- ▶ You may use your preferred Java editor or IDE, such as Eclipse, IntelliJ IDEA, SlickEdit or any other tool. There may be some plugins available for these tools that may assist you in migrating applications between different vendor runtimes. Run a Google search to find out the latest information.
- ▶ If, for any reason, you decide not to use the IBM provided migration plugin (perhaps it does not support your source environment), you may want to write XSLT stylesheets to transform XML deployment descriptors of one vendor into the format of another vendor. One example of such an approach is described in this article:

<http://www.onjava.com/pub/a/onjava/2005/03/09/ejb-migration.html>

To facilitate writing the XSLT or even to generate it automatically, you may want to use Rational Application Developer V6 provided XML plugin or any equivalent XML tool as long as it automates the generation of XSLTs and supports XSL debugging. An example of such a tool is XMLSpy or the Eclipse plugin for XML called XMLBuddy Pro.

Another important issue is the need for maintaining the history of your changes in a version control system. If you are planning to use Rational Application Developer V6, make sure that your version control system is supported, like Rational ClearCase® or CVS, or that you can make Ant scripts to put and retrieve the changes in your version control system. For further information about the supported version control systems, refer to *Rational Application Developer V6 Programming Guide*, SG24-6449.

### 3.3.6 Reviewing the runtime environment

As mentioned before, there is more than one environment to migrate, and you have to review the current assets to ensure that the existing requirements are adequate for WebSphere Application Server V6. At this stage, it is important to leverage all the information related to security requirements in those environments, such as authorized users, user rights, disk quotas and firewall configurations.

### 3.3.7 Reviewing the current build and deployment processes

The procedure to build and deploy your applications will be related to your current application server requirements. The existing scripts to build and deploy your applications have to be reviewed and changed. The same will be needed with the scripts used to manage the application servers. In this book, we have used several different build tools:

- ▶ Ant-driven build
- ▶ Maven build
- ▶ IDE-driven build (in our case, Rational Application Developer V6 build)

### 3.3.8 Assessing current skills

Application developers, architects and administrators can take further advantage of WebSphere Application Server V6 and Rational Application Developer V6 functionalities if they are familiar with those.

Rational Application Developer V6 is a very powerful tool that can ease the development process, so it is a good idea to train your staff in the tool.

Managing WebSphere Application Server V6 could be very different from managing your current application server. While the main concepts are similar, the details will be different. It may take time for your operators to learn and understand how to configure and manage WebSphere Application Server V6.

As mentioned in the previous section, the scripts used to manage the application server will need to be rebuilt, and this have to be done by your operators staff.

### 3.3.9 Reviewing time constraints

Not all the applications can be migrated at the same time. Complexity of the migration, ongoing requirement changes or business needs are constraints to deal with in your planning. Gather the information about the next releases of your applications, and try to put an intermediate delivery of the migrated application release or a late delivery of the migrated application with the new features, but avoid performing the migration and code changes at the same time.

### 3.3.10 Creating a detailed migration plan

All the previous steps will help you to create a detailed plan for your migration. It is important to have a detailed plan including all the previous tasks, with times in accordance to your applications and business needs.

Instead of a migration project, you might have a migration program planned, with as many migration projects as you need, including one or more applications in each project, starting with the less critical ones. This approach will give you the capability to rectify the plans of the projects involving the most critical ones with which you will gain more experience. You might need to consider getting external, experienced help at this point.

## 3.4 Migration alternatives

The migration is done by first setting up the source environment and then modifying the application source files, deployment descriptors and properties so that the application runs in the destination environment. For different applications, there may be more than one way to migrate. This is because developers have choices for migration and build tools, Integrated Development Environments (IDEs), persistence frameworks, databases, etc.

Figure 3-1 on page 52 shows the different migration options that were considered for two of our sample applications: ivata groupware and xPetstore Servlet. Please note that both applications used XDoclet. If your application does not use XDoclet, you will likely consider Alternatives 1 and 4 (we used #4 for the Trade application). If you decide to refactor your application and add XDoclet or WRD tags to your Java code to avoid manual writing of multiple artefacts, then you may also consider Alternatives 2 and 3 even if your original application had no annotation programming.

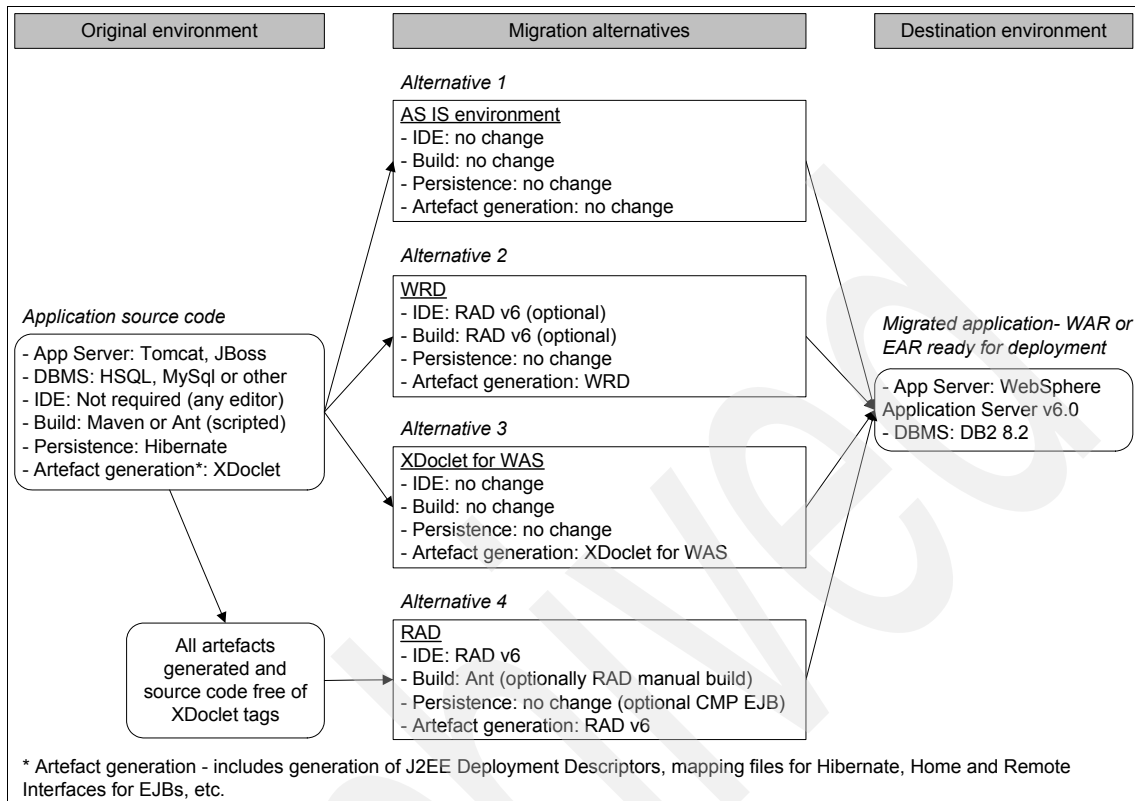


Figure 3-1 Migration alternatives for XDoclet applications

### 3.4.1 Alternative 1: Keeping the existing development environment

This alternative suggests using exactly the same build and development environment for migration and future application development as was used to build the original application. This includes minimal code changes and trying to preserve all frameworks and tools that are built into the application. This will only work if all artefacts generated by XDoclet (such as web.xml and other files) are fully J2EE-compliant. If you need to write WebSphere-specific deployment descriptors, you can use XDoclet merge directories to add content to generated descriptors. WebSphere-specific deployment descriptors can be written by hand or regenerated by Rational Application Developer V6 whenever needed and then moved into the project structure, so you can continue working without using IDE.

Variation to this alternative is Alternative 3: Using XDoclet support for WebSphere.

Reasons for using this option include the following.

- ▶ Take advantage of the existing skills and established processes.
- ▶ Avoid WebSphere-specific extensions and features.
- ▶ Use of XDoclet allows you to code the J2EE application once and then generate deployed files for different vendor runtimes. There is still a need to keep multiple vendor-specific tags in the source code, but this is within the same file.
- ▶ No need to learn new development tools.
- ▶ No need to change the build process and project directory structure.
- ▶ By doing the development by hand or using XDoclet, it is easier to control what gets generated, rather than using IDE wizards that may generate vendor-specific information and proprietary extensions.
- ▶ Version control with annotation-based programming is much easier than with the traditional J2EE approach of editing multiple files when working on a single component.

Reasons for not using this option include:

- ▶ Cannot take advantage of productivity features built into Rational Application Developer V6, such as rapid application development wizards, specialized editors for deployment descriptors, visual development for Struts, JSF and JSPs, built-in automatic publishing into WebSphere Test Server, Universal Test Client, automatic generation of JUnit test cases and many more features that speed up application development.
- ▶ Open source frameworks and tools used to build the application that are popular today may not be popular or even available or supported in one or two years and will require migration at that point.
- ▶ Frameworks used in the application may or may not be flexible enough to take advantage or be extended to support new technologies that might emerge in the future.

### 3.4.2 Alternative 2: Migrating XDoclet tags to WRD

This alternative is somewhat similar to Alternative 1: Keeping the existing development environment, but instead of using XDoclet tags to generate deployment descriptors, Remote and Home interfaces and other artefacts, it suggests replacing most of them with WRD tags; for further details on annotation-based programming, refer to 2.3.4, “WebSphere Rapid Deployment” on page 26.

As far as the choice for the IDE and build, this can be flexible. Rational Application Developer V6 supports code assist for WRD annotations, so it may be a good choice for IDE, but depending on the build structure of the original

project, you may prefer to use Maven or Ant for the build, or perhaps run the build from within IDE (not likely for complex projects since most developers prefer repeatable builds, not manual ones). To facilitate this, Rational Application Developer V6 supports Ant from within an IDE.

Reasons for using this option include:

- ▶ Similar benefits as the annotation-based programming described in the previous alternative (except for the build; see limitations below).
- ▶ If you are using Rational Application Developer V6, you also have code assist for WRD tags.
- ▶ WRD tags are mostly compatible with XDoclet `@web` and `@ejb` tags, so you may be able to maintain your application source code (not binary) portability between different application server runtimes.
- ▶ The current draft of J2EE 1.5 is directed to use new Java 1.5 annotations and the use of WRD sets you in the right direction. It is possible that IBM may provide some kind of migration utility in the future to upgrade your source code from current WRD annotations to J2EE 1.5 annotations when they become finalized and supported by the WebSphere Application Server V6.

Reasons for not using this option include:

- ▶ At the time of this writing, WRD is still very new (released for the first time) and some of the code generation is not working properly. We have found some errors in generated code for specific scenarios and opened PMRs with IBM support to resolve those issues.
- ▶ Annotation-based programming is supported by WRD (there is more to WRD than just annotations), but at the time of this writing, WRD is not as extensive as XDoclet. For example, it only supports annotations specific to Web services, EJBs and Servlets and other J2EE artefacts, but has no support for Hibernate or other frameworks (XDoclet has tags and plugins for many different frameworks). Therefore, there may be some need to use XDoclet tags for the generation of selected artefacts. For a full list of the XDoclet-supported extensions, visit the following URL:  
<http://xdoclet.sourceforge.net/xdoclet>
- ▶ It may be challenging to use WRD as part of your custom Ant or Maven build scripts, since IBM does not provide Ant tasks for WRD (at least not in the current version). In our migration example, we have used Rational Application Developer V6 as our IDE and build tool.
- ▶ WRD does not fully support variable substitution, like XDoclet does. This is likely to be improved in the next version (today, only limited support is available, but not for JNDI names). Here is an example of a XDoclet tag that is not supported by WRD.

#### Example 3-1 XDoclet tag

---

```
/**
 * @ejb.resource-ref
 *   res-ref-name="${jndi.queue.ConnectionFactory}"
 *   res-type="javax.jms.QueueConnectionFactory"
 *   res-auth="Container"
 *   jndi-name="${orion.queue.ConnectionFactory}"
 */
```

---

- ▶ WRD templates are not customizable in the current version, so if you do not like what is generated, you cannot change it.

### 3.4.3 Alternative 3: Using XDoclet support for WebSphere

This is quite similar approach to Alternative 1: Keeping the existing development environment, but instead of using XDoclet tags from the original application, this needs to be extended with WebSphere-specific XDoclet tags. This is especially important for EJB doclets and generating WebSphere-specific deployment descriptors. For further information about XDoclet tags for WebSphere, visit the following URLs:

<http://xdoclet.sourceforge.net/xdoclet/tags/ibm-tags.html>

<http://xdoclet.sourceforge.net/xdoclet/ant/xdoclet/modules/ibm/websphere/web/WebSphereWebXmlSubTask.html>

Reasons for using this option include:

- ▶ All of the same reasons as in Alternative 1: Keeping the existing development environment.
- ▶ This alternative allows you to take advantage of some WebSphere-specific XDoclet tags and avoid manual writing of the deployment descriptors and having to put them into XDoclet merge directories, thus allowing for a more complete annotation programming and better developer experience.

Reasons for not using this option include:

- ▶ XDoclet may not support the latest version of WebSphere or it may lag behind in time. At the time of writing this book, the latest XDoclet module for WebSphere supports version 5.1, while this book covers WebSphere Application Server V6. These two versions are compatible, but you may not be able to take advantage of some of the features or may be forced to write some deployment descriptors by hand and keep them outside of the Java source, thus not getting the full benefit of annotation-based programming.
- ▶ Existing XDoclet tags for WebSphere only support Top Down mapping for CMP EJBs. If you need Bottom Up or Meet in The Middle mapping, you will have to use Rational Application Developer V6 to generate those mappings

each time you have a new database schema or change persistent field of your CMPs. Once generated, you can include those mappings into your Ant or Maven build.

- ▶ The same limitations as in Alternative 1: Keeping the existing development environment still apply here.

### **3.4.4 Alternative 4: Developing using Rational Application Developer V6**

This alternative will include removing all XDoclet annotations for deployment descriptors and editing these directly via advanced DD editors built into Rational Application Developer V6. This is also likely to force change in the project directory structure, build process and/or scripts, source code and deployment descriptors, but you may keep frameworks used in the original application if you wish.

In some cases, we have seen that cost is considered to be a disadvantage in Rational Application Developer V6 compared to Eclipse or other free tools. On the other hand, better tools translate into higher development productivity and a faster return on investment.

Reasons for using this option include:

- ▶ If you decide to follow the path of using the IDE for your development, there is no doubt that Rational Application Developer V6 will be your best choice for WebSphere Application Server V6. This is because of many integration points between these two products, such as local and remote deployment, debugging, configuration, Universal Test Client, specialized editors for J2EE generic and WebSphere-specific deployment descriptors for EJB, Web, Web services, and many more capabilities that allow a very productive development.
- ▶ We hope that you use Eclipse or another advanced Java development tool for all alternatives, but there are many useful features and plugins available in Rational Application Developer V6 in addition to Eclipse. These include advanced XML editors, database tools, UML modelling, excellent JSP and JSF development tools, version control integration, and too many others to mention. All of this should increase your development speed.
- ▶ Even if you are using Rational Application Developer V6, you can still use Ant or Maven for your build process.

Reasons for not using this option include:

- ▶ To take full advantage of Rational Application Developer V6, you may have to change your project directory structure. This may be time consuming. The



good news is that Rational Application Developer V6 is based on Eclipse, is very flexible and can likely adapt to your current directory structure with minimum work on your part.

- If you use XDoclet to generate your artefacts, or you use other frameworks, and you import a perfectly working project into Rational Application Developer V6, you may see a lot of error and warning messages. This is because the validation options in this IDE are not able to verify your file links or class dependencies. It may take significant effort to fix those errors and messages, or you may choose to disable some validators to remove those messages. See Figure 3-2 for an example.

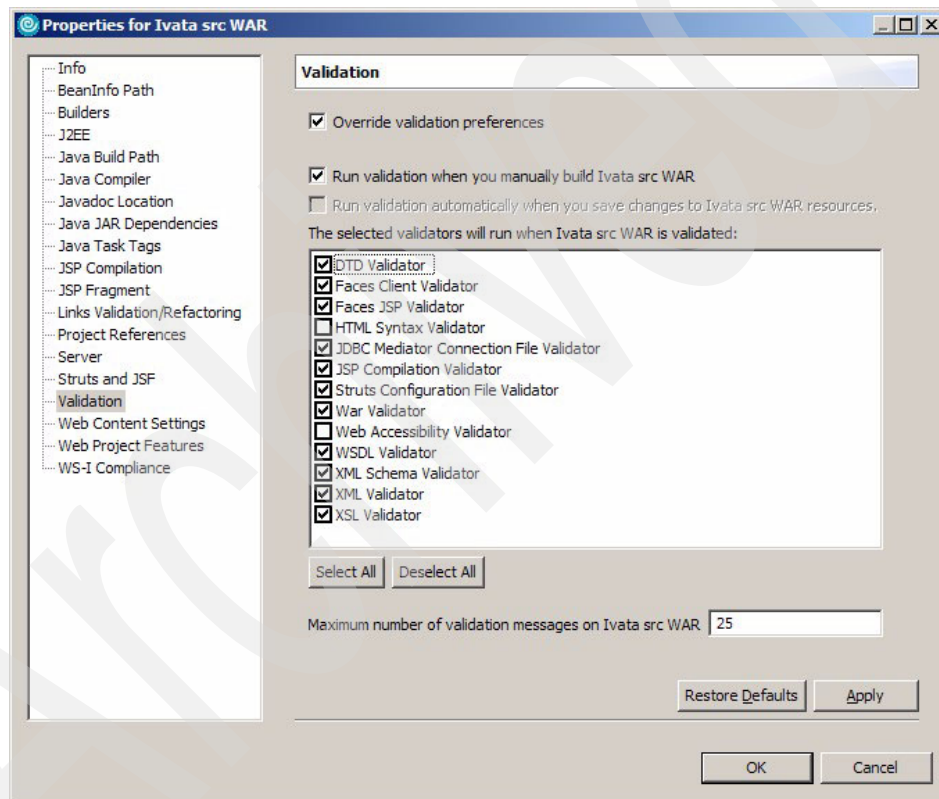


Figure 3-2 Project validation preferences in Rational Application Developer V6

A variation on this alternative is to replace your current persistence model (whether you use Hibernate, JDO or JDBC) with CMP EJBs. This is likely to cause additional work and may involve a steep learning curve. There are benefits of performing application development with Sessions and Entity EJBs, as is proven by many production projects, yet some authors argue against EJBs. This

may be a decision, that is more religious than technical. We suggest that you keep your options open and use the right tools for the right task.

## **3.5 Migrating the site**

As discussed in 3.1.2, “Migrating the different environments” on page 32, there are three kinds of environments to migrate. In this section, we will only talk about the runtime environments, such as testing, staging and production. There are basically three ways to migrate these kinds of environments.

### **3.5.1 Switchover migration**

In the switchover migration scenario, a separate computer (or cluster) is needed to install the new environment and then perform the migration. The installation and migration is performed while the original environment is still in production, keeping to a minimum the production server downtime at the time of switching over to the newly migrated environment.

Since the original environment was in production until the last minute, it is up to date and can be used as the production environment backup if anything goes wrong during the switchover.

### **3.5.2 Coexistence migration**

If your current set of server machines is powerful enough to run two application servers, this migration scenario could be a viable alternative. Pay special attention to the ports used by every application server, and configure the new server to use different ports from the previous installation. In this way, you can have your current installation running while you configure the new application server, not needing a second server.

Compared with the switchover migration, this approach has a higher potential risk of affecting the current production environment since both the original and the new environments reside on the same physical machine.

### **3.5.3 Offline migration**

This migration scenario has the maximum downtime. The runtime environment has to be non-operational for the duration of the migration since the idea of this migration scenario is to replace the current environment with the new one, in the same server. Typically, this scenario will be used when the server is hosting a non-critical application or there is no additional hardware available.

### 3.5.4 Database migration

If you plan to migrate the database, consider performing an initial database migration to evaluate the time and effort. Database migration can be a simple task or be even more difficult than the application migration. Several factors will affect the database migration; some of them are database size, design and maintenance.

There are several different tools available for the database migration:

- ▶ The IBM DB2 Migration Toolkit helps you migrate from Oracle (versions 7, 8i and 9i), Sybase ASE (versions 11 through 12.5), Microsoft SQL Server (versions 6, 7, and 2000), Informix (IDS v7.3 and v9), and Informix XPS (limited support) to DB2 UDB V8.1 and DB2 V8.2 on Windows, UNIX and Linux and DB2 iSeries including iSeries v5r3. The DB2 Migration Toolkit is available on a variety of platforms including Windows (2000, NT 4.0 and XP), AIX, Linux, HP/UX and Solaris, for further details visit the following URL:  
<http://www.ibm.com/software/data/db2/migration/mtk>
- ▶ Migration from MySQL to DB2 is covered in the redbook *MySQL to DB2 UDB Conversion Guide*, SG24-7093-00.
- ▶ Inspirer Systems Ltd. offers the SQLWays migration tool; the demo version is available for download and has a limit of seven characters in the length of column names.  
<http://www.ispirer.com/products>
- ▶ MySQL database has the capability to export to other database formats. For more information, see this blog posting at the following URL:  
[http://www.logemann.org/blojsom/blog/default/2005/05/10/why\\_i\\_love\\_MySQL\\_as\\_developer\\_database.html?page=comments](http://www.logemann.org/blojsom/blog/default/2005/05/10/why_i_love_MySQL_as_developer_database.html?page=comments)
- ▶ If your project is using Hibernate, then you can use Hibernate to migrate to a new database, since Hibernate can generate the DDL for the specific database dialect. For example, xPetstore Servlet DDL is generated with the task `ant xdoclet.hibernate.ddl`.

## 3.6 Testing the migrated site

In migrating servers as in developing, testing is one of the most important activities in the project. Migration is not finished until the new server site accomplishes all the test cases. This means that you have to migrate the test cases too. If you do not have test cases, consider creating them and account for them in your plan.

As you test the migrated site, you may need to go back to previous steps due to errors found during or after migration, and fix them. Therefore, you need to assign enough time not only to test the site, but to test it again if anything goes wrong the first time.

Many of the same principles of version-to-version migration apply to the vendor-to-vendor migration. For further information about version-to-version migration, refer to *WebSphere Application Server V6 Migration Guide*, SG24-6369-00.

## 3.7 Going live

Once you have migrated and successfully tested your new environment, it is time to put it in production. Whichever migration scenario you choose, you will likely need to go offline to stop the transactions on your database and possibly migrate its data.

In some cases, it may be possible to deploy the new application against the live production database, run production workload in parallel by spraying requests across two instances of the application running in different environments and, once you are sure it works well, you can shut down your old environment and handle the entire workload with the new runtime.

Building highly available systems is a big topic. Refer to *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392-00 for further details.

## 3.8 Naming conventions

During the migration examples, we will refer to different environments and directories by their names. In this section, we will establish those names and explain their meaning. Your migration plan should include a section explaining this concept, to ensure that all the people involved in the project are using the same terminology to refer to the same concepts, easing the communication.

Some terms used in the book are:

- ▶ Source environment

The source environment is where the original application is running. This includes the application server, database manager, directories, and so forth.

- ▶ Destination environment

The destination environment is where the application is going to be migrated. This includes the application server, database manager, directories, and so forth.

► <rad\_home>

This is the home directory where Rational Application Developer V6 is installed.

► <rad\_workspace>

This is the directory where we choose to place the workspace to work with Rational Application Developer V6.

► <was\_home>

This is the home directory where WebSphere Application Server V6 is installed.

► <profile\_name>

This is the name of the WebSphere profile used to deploy the application.

► <db2\_home>

This is the home directory where DB2 Universal Database V8.2 is installed.

► <source\_home>

This is the home directory where the source files of the application being migrated are located.

► <weblogic\_home>

This is the home directory where BEA WebLogic Server 8.1 is installed.

► <jboss\_home>

This is the home directory where JBoss 3.2.7 is installed.

► <tomcat\_home>

This is the home directory where Apache Tomcat 5.5.9 is installed.



# Installation and configuration

In this chapter, we provide an overview of the environment used in the migration examples, describing the hardware and software utilized.

We also explain how to install and configure the IBM products used in the migration. This is not intended to be a detailed installation guide; it does, however, provides a step-by-step description of how we installed the products, allowing you to follow the migration examples explained in this book.

This chapter is organized in the following sections:

- ▶ Introduction
- ▶ Hardware and software installed
- ▶ Rational Application Developer V6
- ▶ WebSphere Application Server V6
- ▶ WebSphere Administrative Console
- ▶ DB2 Universal Database V8.2
- ▶ Installation directories

## 4.1 Introduction

This section describes the environment that was built as a destination environment for all the examples presented in this book. Here you will find detailed instructions to install all the required software, as well as a description of the environment where it was installed. The installation instructions presented describe the installation we performed, and are not intended to constitute an installation guide for every migration scenario.

In 4.5, “WebSphere Administrative Console” on page 78, we also describe some basic administration steps to manage the application server and allow you to get used to the Administration Console.

Finally, in 4.8, “Installation directories” on page 87, you will find a summary of the directories where the applications were installed during this chapter.

For detailed installation instructions and prerequisites, refer to the product documentation. “Related publications” on page 325 provides a list of additional literature and online resources.

## 4.2 Hardware and software installed

The following list highlights the hardware and software we used to build our ITSO migration labs.

- ▶ IBM NetVista with Pentium® IV 3.06 GHz processor, 2 GB RAM and a 37 GB HDD
- ▶ Microsoft Windows 2000 Professional with Service Pack 4 Build 2195
- ▶ Microsoft Internet Explorer V 6.0.2800.1106
- ▶ IBM Rational Application Developer V6
- ▶ IBM WebSphere Application Server V6
- ▶ IBM DB2 Universal Database V8.2, Express Edition

## 4.3 Rational Application Developer V6

In this section, we show how we installed and configured Rational Application Developer V6. Here, we are installing our WebSphere Application Server as an integrated test environment, so we do not have to install it separately. This way, you can start and stop your application server from Rational Application Developer V6, as well as deploy applications and debug the applications running on it.



You can download a trial Rational Application Developer V6 at the following URL:

<http://www.ibm.com/developerworks/downloads/r/rad>

The following list provides the high-level steps we used to install our Rational Application Developer V6 environment.

1. Start the Rational Application Developer V6 setup program called `launchpad.exe`.
2. Select the option **Install IBM Rational Application Developer V6**.
3. Click **Next** in the welcome window.
4. The license agreement window appears. Select **I accept the terms in the license agreement** and click **Next**.
5. Choose an installation directory. We installed Rational Application Developer V6 in the following directory  
`C:\IBM\Rational\SDP\6.0`  
From now on, we will refer to this directory as `<rad_home>`.
6. From the Features to install selection window, we only select the **IBM WebSphere Application Server V6 Integrated Test Environment** and click **Next**.
7. A confirmation window appears with a description of the products and features to be installed. Verify the summary information and click **Next** to start the installation.
8. A confirmation window will appear, informing you that Rational Application Developer V6 has been successfully installed. Click **Next** to continue.
9. Deselect the Agent Controlled installation and then click **Finish**.

At this point, Rational Application Developer V6 is installed and WebSphere Application Server V6 is also installed as an integrated test environment. We will use the very same JVM provided with WebSphere Application Server V6 installation for running non-IBM command line tools, such as Maven or Ant.

In order to allow command line tools to use the JVM installed with WebSphere Application Server V6, we need to define the `JAVA_HOME` in the system environment variables and point it to the directory where the JVM is installed. In our scenario, we set the `JAVA_HOME` variable as follows:

```
JAVA_HOME=C:\IBM\Rational\SDP\6.0\runtimes\base_v6\_jvm
```

Now, we can run some Java command line tools directly from any command line window.

### 4.3.1 Installing the interim fix

Once you have installed Rational Application Developer V6, you can look for new updates using the Rational Product Updater. You can install the fixes online or offline. The offline installation is recommended when you have more than one installation to update because you download it once, and can then install it as many times as you need. The offline installation requires an additional configuration step to run. The following section describes the steps to update Rational Application Developer V6 either online or offline.

#### Online update

To install the Rational Application Developer V6 interim fix online, follow these steps:

1. Start the Rational Product Updater by choosing **Start** → **Programs** → **IBM Rational** → **Rational Product Updater**.
2. In the first window, you will see all the Rational products installed. Click **Find Updates** to look for all the available updates for those products.
3. If you do not have a fix installed, you might need to install a fix for the Rational Product Updater first. You will be notified when you perform the first install, as depicted in Figure 4-1. Click **OK** to install the necessary updates and proceed with the update procedure.

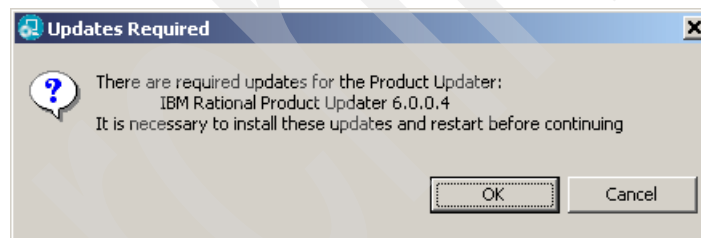


Figure 4-1 Required updates

4. In the main window, you will see all the available updates for all your installed products. Check the fixes to install and click **Install Updates**.
5. A licence agreement window will appear. Accept the license agreement and click **OK**. Depending on your connection speed, this operation could take several minutes.
6. Once the updates are installed, the main window will show all the updates installed.
7. You can now exit the Rational Product Updater.

## Offline update

The steps to perform an offline update are basically the same as for the online update, but you have to download the updates first.

You can download the last updates from the following URL:

<http://www.ibm.com/software/support>

Here are the steps to follow when performing an offline update:

1. Once you have downloaded the updates, unpack the file(s) in a directory. For this operation, we used the following directory:  
C:\temp
2. Start the Rational Product Updater by choosing **Start** → **Programs** → **IBM Rational** → **Rational Product Updater**.
3. In the directory where the files were decompressed, you will find an XML file. Modify the XML file as explained in the instructions of the downloaded updates, replacing the text between square brackets with the path used to decompress the files. Your modified XML file should look like the one shown in Example 4-1.

*Example 4-1 Rational Product Updater xml configuration file*

---

```
<?xml version="1.0" encoding="UTF-8"?>
<update-policy>
  <url-map pattern="com.ibm.rational.application.developer.update"
url="file:/C:/temp/rad/60/update/site_002.xml"/>
</update-policy>
```

---

4. Once you have modified the XML file, go back to the Rational Product Updater and select **Preferences** → **Update sites**. Click the **Browse** button and select the file you have already modified.
5. Now you have configured the Rational Product Updater to look in the directory you specified for the updates. Follow steps 2 to 7 of the Online update to complete the installation.

### 4.3.2 Testing the application server

Rational Application Developer V6 allows you to manage the WebSphere Application Server V6 from the development environment, but you can also run administration commands from the command line if you like. You will find WebSphere Application Server V6 installed inside of the Rational Application Developer V6 installation directory in the following path:

```
<rad_home>\runtimes\base_v6
```

From now on, when using Rational Application Developer V6 in any of our migration projects, we will refer to this directory as <was\_home>.

WebSphere Application Server V6 uses WebSphere profiles to group user data and configuration. These WebSphere profiles share the same binaries from the WebSphere Application Server V6 installation but can be distinguished from one another by their different home directories. We will explain WebSphere profiles in more detail in 4.4.2, “WebSphere profiles” on page 73 when we cover a full WebSphere Application Server V6 installation. At this point in time, it is only necessary to provide a quick introduction to WebSphere profiles since, by default, a new profile is created during the product installation.

The default WebSphere profile created during the product installation defines a single application server, called *server1*, that runs as a standalone server, and is configured within Rational Application Developer V6. You can use this server, or define another WebSphere profile and server to test and run your applications. For our examples, we just used the default WebSphere profile.

This default WebSphere profile is located in the following directory structure:

```
<was_home>\profiles\default
```

Within this directory structure, the subdirectories we will be referencing the most are:

► bin

In this directory, you will find administrative commands for the profile. System administration commands are now profile-specific, so you must execute the commands from their particular profile/bin directory.

► logs

Inside the logs directory, you will find one directory per server containing the log files of each. The SystemErr.log and SystemOut.log files are inside of this directory, as well as the serverStatus and the trace log files. So if you need to find the logs for server1, you have to go to the following directory:

```
<was_home>\profiles\default\server1
```

► config

Within this directory, you will find all the configuration files.

## Verifying the configuration

Inside of the Rational Application Developer V6, using the Servers view, you will find the defined server referenced as:

```
WebSphere v6.0 Server @ localhost
```

If you double-click this, a new view will appear with the server overview where you can verify the settings of the server.

## Starting the server

You can start the server using the Rational Application Developer V6 interface or the command line. To start the application server from the command line, run the **startServer** command located in the following directory:

```
C:\ibm\Rational\SDP\6.0\runtimes\base_v6\profiles\default\bin
```

The **startServer** command must receive as a parameter the server name specified as follows:

```
startServer server1
```

To start the server from Rational Application Developer V6, go to the Server view, right-click the server you want to start and then select **Start**.

## Verifying the server status

If you are running Rational Application Developer V6, you can see the server status in the Servers view, in the Status column, as shown in Figure 4-2.

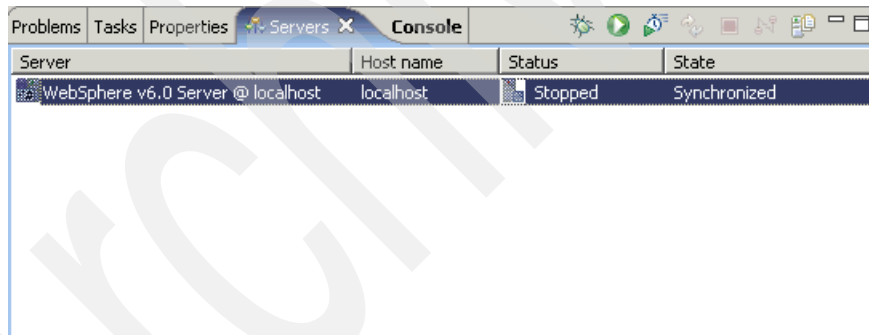


Figure 4-2 Server View in Rational Application Developer V6

If you are not using Rational Application Developer V6 or prefer to use the command line, you can run the following command:

```
serverStatus server1
```

Alternatively, you can run the same command with the following parameter to verify the status of all the servers for that specific profile.

```
serverStatus -all
```

**Note:** When using WebSphere Application Server V6 - Base, you will only be able to configure one application server for each profile, but you can still create as many profiles as you need. When using WebSphere Application Server V6 - Network Deployment, you can create several application servers within the same profile. In the Network Deployment version, you can still create multiple profiles containing multiple application servers.

### Stopping the server

Similarly, as with starting the server, you can stop the server using the Rational Application Developer V6 interface or the command line. To stop the application server from the command line, run the **stopServer** command located in the following directory:

```
C:\IBM\Rational\SDP\6.0\runtimes\base_v6\profiles\default\bin
```

The **stopServer** command must receive as a parameter the server name specified as follows:

```
stopServer server1
```

To stop the server from Rational Application Developer V6, go to the Server view and right-click the server you want to start. Then select **Stop**.

## 4.4 WebSphere Application Server V6

Although Rational Application Developer V6 includes WebSphere Application Server V6 as an integrated test environment, some migration examples we will cover later in this book do not use Rational Application Developer V6 as part of the migration process and only make use of WebSphere Application Server V6.

This section describes the steps to install WebSphere Application Server V6 - Base in the same system where Rational Application Developer V6 was installed. These steps will differ slightly from an installation in an environment without Rational Application Developer V6, because we will have two different WebSphere Application Servers in the same environment, potentially using the same ports. The installation wizard will recognize the existing installation and propose alternative ports, avoiding this kind of conflict.

**Note:** WebSphere Application Server V6 allows you to have multiple installations of the same, and previous, WebSphere Application Server releases in the same system. For details about installation options and topologies, refer to *WebSphere Application Server V6 Planning and Design*, SG24-6446.

Before you start, make sure that you will perform the installation with a user that belongs to the *Administrators* group, and having the following advanced user rights:

- ▶ To act as part of the operating system
- ▶ To log on as a service

This is a requirement to ensure the correct installation of WebSphere Application Server V6.

You can download a trial WebSphere Application Server V6 at the following URL:  
<http://www.ibm.com/developerworks/downloads/ws/was>

The following list provides the high-level steps we performed to install our WebSphere Application Server V6 environment.

1. Start the setup program. A new window will appear, showing a Welcome message. Click **Next**.
2. When the licence window appears, accept the terms of the license agreement and click **Next**.
3. A new window will appear, informing the successful check of the system requirements. Click **Next** to continue.
4. At this point, the installation program will inform you of the detection of an existing copy of WebSphere Application Server V6 on the computer, allowing you either to install a new copy of WebSphere Application Server V6 or to add features to the existing options. Select **Install a new copy of the V6 Application Server product** and click **Next**.
5. An information window will appear, giving instructions about the installation of a second copy of the WebSphere Application Server V6. Click **Next**.
6. Choose an installation directory and click **Next**. We installed WebSphere Application Server V6 in the following directory.  
C:\IBM\WebSphere\AppServer
7. A window with the features to install will appear. Leave all the options checked, and click **Next**.
8. A new window appears, allowing you to define the ports for the new copy of WebSphere Application Server V6. The installation program will detect the previous installation, and will propose a new value for the port configuration, as well as a reference for the default value, as depicted in Figure 4-3 on page 72. You can change the ports if they conflict with another application. We accepted the proposed ports and clicked **Next**.
9. Specify a node name and host names. In our installation, we accepted the default value for the node name. Click **Next**.

10. In the following window, you can configure WebSphere Application Server V6 to run as a service. You must provide a user and password to log on the service, and click **Next** to proceed with the installation.
11. A confirmation window appears with a description of the products and features to be installed. Verify the summary information and click **Next** to start the installation.
12. A confirmation window will appear, informing you that the WebSphere Application Server V6 has been successfully installed. Deselect the **Launch the first steps console** option and then click **Finish**.

The values in the following fields define the ports for the Application Server and do not conflict with other profiles in this installation. Another installation of WebSphere Application Server or other programs might use the same ports. To avoid run-time port conflicts, verify that each port value is unique.

Administrative console port (Default 9060):	9061
Administrative console secure port (Default 9043):	9044
HTTP transport port (Default 9080):	9081
HTTPS transport port (Default 9443):	9444
Bootstrap port (Default 2809):	2810
SOAP connector port (Default 8880):	8881
SAS SSL ServerAuth port (Default 9401):	9404
CSIv2 ServerAuth listener port (Default 9403):	9405
CSIv2 MultiAuth listener port (Default 9402):	9406
ORB listener port (Default 9100):	9101
High availability manager communication port (Default 9353):	9354
Service Integration Port (Default 7276):	7277
Service Integration Secure Port (Default 7286):	7287
Service Integration MQ Interoperability Port (Default 5558):	5559
Service Integration MQ Interoperability Secure Port (Default 5578):	5579

Figure 4-3 WebSphere Application Server V6 port configuration

#### 4.4.1 Installing the latest fixpack

Once you have completed the installation of WebSphere Application Server V6, you need to check for new updates of the product. You can find the latest updates of the WebSphere Application Server by clicking the following URL:

<http://www.ibm.com/software/webservers/appserv/was/support>



At the time of writing, the latest fix available was the Refresh Pack 6.0.1, so the following steps will describe how to install that fix. These steps are basically the same for every installation, but may not be exactly the same for future versions. Nevertheless, you can find detailed instructions to install the fixpack in the same URL where you found the fix.

1. Download the fixpack and unpack the file in the same directory where you have installed WebSphere Application Server V6. You may see a new directory called updateinstaller in your <was\_home>.
2. Run the **update.exe** command in the <was\_home>\updateinstaller directory.
3. After a few moments spent looking for installed software, a welcome page appears. Click **Next**.
4. In the next window, select which installation of WebSphere Application Server you want to update. We selected:  
`C:\IBM\WebSphere\AppServer`  
since this is the directory where we install WebSphere Application Server V6. Click **Next**.
5. A new window appears in which you have to select the maintenance operation. Select **Install maintenance package** and click **Next**.
6. At this point, you have to select the maintenance package to install. Leave the default setting and click **Next**.
7. A new window appears, informing you that the maintenance package requires an update to the JDK that is already in use by the installer itself, so the JDK has to be copied to another location. Click **Next**.
8. A new window appears, informing you of the success of the previous operation. You have to click **Relaunch** to restart the update process, this time with the right JDK.
9. The setup application will check again for the installed software, and you have to select the maintenance package to install. Once again, leave the default setting and click **Next**.
10. A confirmation window will appear, showing the products to be updated and the maintenance level that will be applied. Click **Next** to start the installation.
11. After a few moments, a new window appears, informing you that the maintenance package was installed successfully. Click **Finish** to close the Update installer.

#### 4.4.2 WebSphere profiles

Before we start working with WebSphere Application Server V6, we need to understand the concept of WebSphere profiles. We already mentioned

WebSphere profiles in 4.4, “WebSphere Application Server V6” on page 70; in this section, we will explain this concept in more detail.

WebSphere Application Server V6 has the ability to run multiple instances on a single machine using a single WebSphere installation. These instances of the WebSphere Application Server are called WebSphere profiles.

When you install WebSphere Application Server V6, the product files have two components:

- ▶ A set of shared read-only product static files or product binaries to be shared by any functional instance of the WebSphere Application Server product.
- ▶ A user data set of configurable files that are customized. These files are called WebSphere profiles. User data includes WebSphere Application Server configuration, application servers, installed applications, properties, logs, and so on.

Each profile has a separate configuration but still uses a single installation of the WebSphere binaries. Each profile is distinguished by its base path, its own directory structure, and its own `setupCmdLine` script to configure its command line environment.

WebSphere profile support provides:

- ▶ The ability to create different user data (WebSphere profiles) and run multiple instances of WebSphere Application Server sharing the product binaries.
- ▶ WebSphere profiles that define a functional instance of the server.
- ▶ The ability to have multiple profiles running on the same shared product binaries.
- ▶ Profile templates that you can use to create different server profiles.
  - Application Server Profile
  - Deployment Manager Profile
  - Custom profile

Advanced users can tweak existing profile templates to create new profile instances.

**Note:** The different types of WebSphere Profiles (Application Server, Deployment Manager and Custom) are only available through WebSphere Application Server V6 - Network Deployment. WebSphere Application Server V6 - Base provides only one type of profiles which is Application Server.

- ▶ A command line tool, wasprofile, that you can use to create additional WebSphere profiles based on templates.

WebSphere Application Server V6 - Base also provides a default application server profile that is created as part of the product install process. This is not the case when you are installing WebSphere Application Server V6 - Network Deployment.

### 4.4.3 Creating new WebSphere profiles

As mentioned previously, WebSphere Application Server V6 allows the creation of different profiles within a single installation. You may want to create different profiles to test and run different applications, and isolate their possible problems during the migration. In our migration examples, we created new WebSphere profiles as we were testing and deploying applications at different stages of the migration.

The following steps show how to create a new WebSphere profile.

1. To create a new profile, we will use the profile creation wizard. You can launch the wizard in two different ways, depending on the software you have installed, that is, having installed WebSphere in addition to Rational.
  - From WebSphere Application Server  
Click **Start** → **Programs** → **IBM WebSphere** → **Application Server v6** → **Profile creation wizard**.
  - From Rational Application Developer V6  
Within the Rational workspace, click in the top menu **Window** → **Preferences** → **Server** → **WebSphere** and click **Create Profile**.

**Note:** You could also create profiles by using the command line profile tool found here: <was\_home>\bin\wasprofile.bat. For more information about profiles in WebSphere, visit the following URL:

[http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tins\\_instances.html](http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tins_instances.html)

2. Once the profile creation wizard starts, click **Next** in the Welcome window.
3. In the next window, enter a name for the profile you are creating. We will use the name ITSOExampleProfile as shown in Figure 4-4 on page 76. Click **Next**.

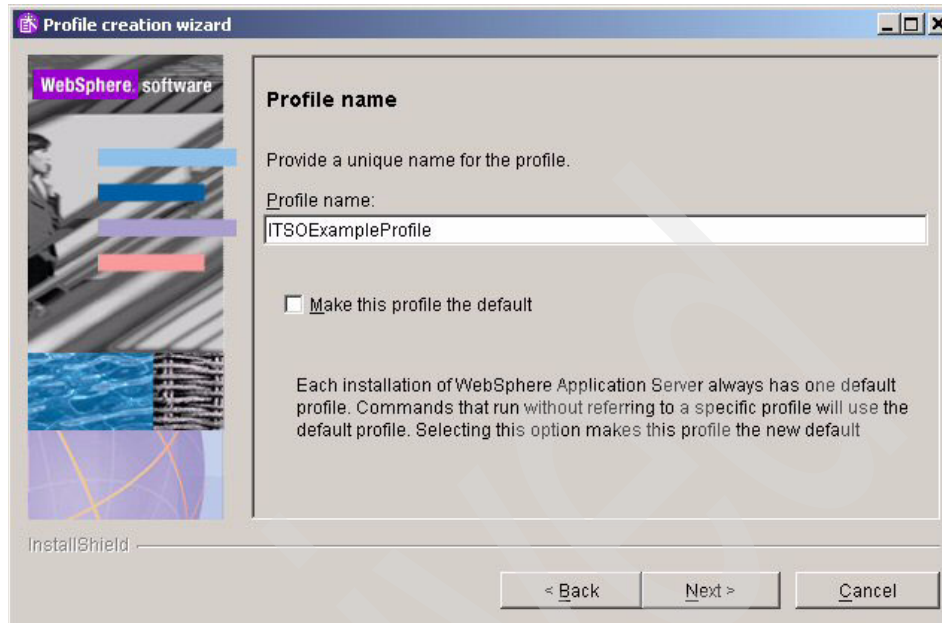


Figure 4-4 Creating a new profile: naming the profile

4. Choose the default directory for the new profile. We selected the default directory <was\_home>\profiles\ITSOExampleProfile. Click **Next**.
5. In the next window, we accepted the default names for our Node and Host names. Click **Next**.
6. In the next window, you will see port assignments for your new profile as depicted in Figure 4-5 on page 77. This is useful because the profile creating tool detects all of your existing profiles and the ports they use and proposes unique port numbers for your new profile. Therefore, we suggest that you check the ports, avoiding problems with any other application using those ports. In our scenario, we accepted the defaults and clicked **Next**.

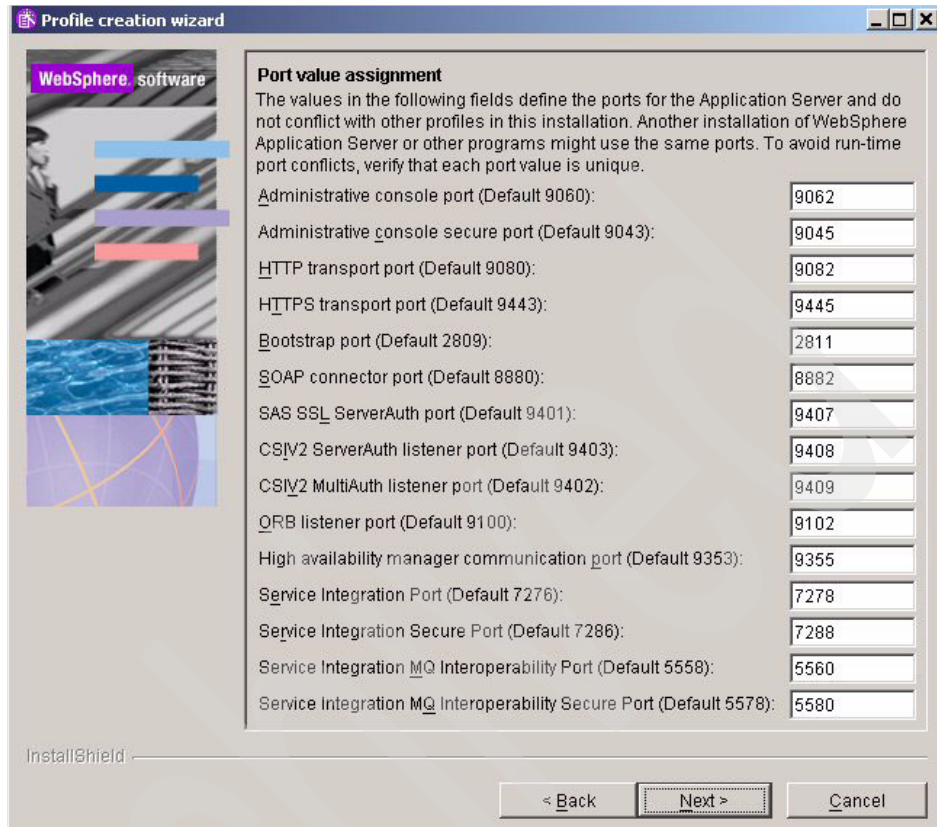


Figure 4-5 Creating a new profile - Port assignment

7. In the Windows Service Definition window, deselect **Run the Application Server as a Windows Service** and click **Next**.
8. In the Profile Summary window, review the values you have entered in previous windows and click **Next**. At this point, the profile creation starts and you may have to wait a few minutes before it is finished. Click **Finish** in the last window.

When you use WebSphere Application Server V6 - Base, you can have a single binary install of the Application Server and you can create multiple profiles. Each profile has a single server defined and the name of that server is always server1. If you were using WebSphere Application Server V6 - Network Deployment, you would be able to create multiple servers with different names in a single profile.

Once you have the product installed and a profile created, you can start the application server from the command line; run the **startServer** command located in the following directory:

```
<was_home>\profiles\ITSOExampleProfile\bin\
```

The **startServer** command must receive as a parameter the server name specified as follows:

```
startServer server1
```

**Note:** Since you will probably create different profiles, you may want to check at a later time the ports assigned for every profile. The ports assigned to a profile are stored in the following file:

```
<was_home>\profiles\<profile_name>\logs\portdef.props
```

Also, you can find the configured ports in the following configuration file:

```
<was_home>\profiles\<PROFILE_NAME>\config\cells\<CELL_NAME>\nodes\  
                                         \<NODE_NAME>\serverindex.xml
```

## 4.5 WebSphere Administrative Console

The Administrative Console is a Web-based interface that provides a graphical interface for managing and configuring your servers. It provides multiple wizards, with entry validation and other facilities and administration aids, translating all the settings in the corresponding configuration files, so you do not have to manually edit any XML configuration files.

From the Administrative Console, you can install, uninstall, start and stop servers and applications. WebSphere Application Server V6 also provides administrative scripts via the wsadmin client; administrative scripts may be the tool of choice, in production environments, for installing and administering applications and servers.

For our migration examples, we will use the Administrative Console for all our administrative tasks.

To launch the Administrative Console, make sure that you have the application server (server1) up and running, then open a Web browser and access the Administrative Console through the following URL:

```
http://localhost:9060/ibm/console
```

The Administrative Console login will appear. At this point, there is no security configured, so you may enter any user name; this name will be used to keep a session with your Web browser.

**Note:** It is important that you log out before closing the Administrative Console, otherwise it may cause a login conflict next time you log in with the same user.

Once you are logged in the Administrative Console, a new window appears as depicted in Figure 4-6. From this window, you will be able to install applications, start and stop servers and applications, configure JDBC providers and datasources and perform all the necessary administrative tasks.

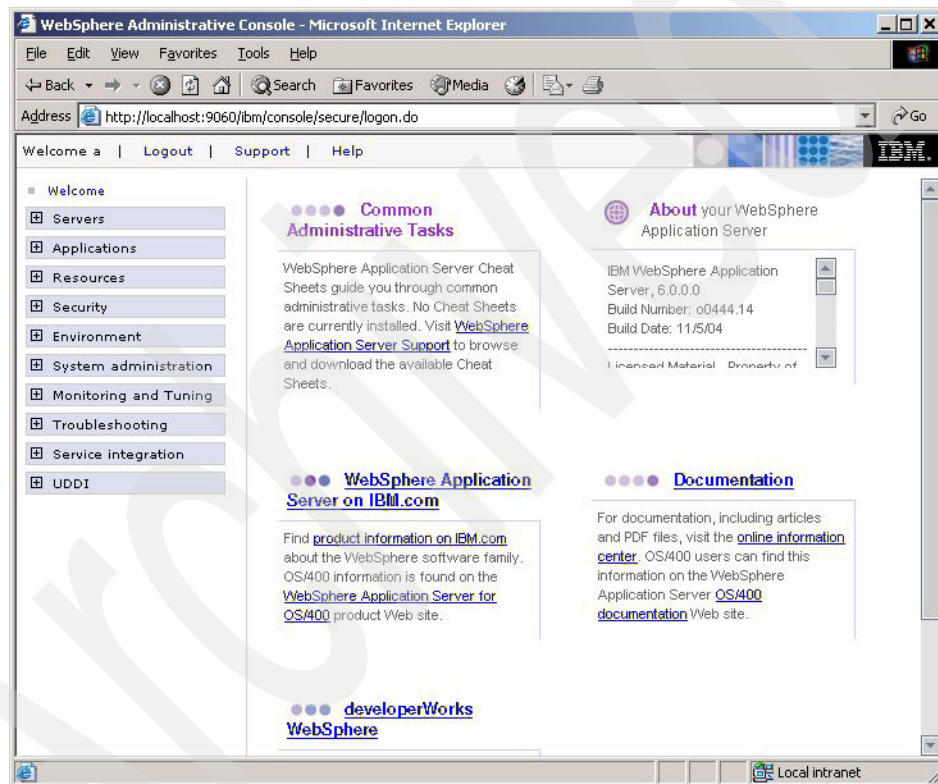


Figure 4-6 WebSphere Application Server V6 Console first window

### 4.5.1 Managing the application servers

In this section, we will provide a high-level overview of the basic administration functions that you can perform using the Administrative Console. The intent of this section is to give you the very first tools to deploy your applications and control the status of the server and the applications. For more information about the administrative operations available in the Administrative Console, refer to

Once you are logged in the Administrative Console, you can manage and configure your server. The most basic operations you can perform inside of the application server are the following.

- ▶ Verifying the server status and configuration.
- ▶ Installing applications.
- ▶ Managing applications.

### **Verifying the server status and configuration**

From the Administrative Console, go to **Servers** → **Application Server**.

A list of the existing servers will appear, showing the name of the server, the node in which it is defined and the version of it. Click **server1** to go to the detailed information of this server.

You should see the configuration of the server, where you can select and change the settings of the EJB container, the Web container and the messaging engines among others.

By clicking the **Runtime** tab, you will be able to see the status of the server and other additional properties.

### **Installing applications**

During the migration and development process, you can install your applications using Rational Application Developer V6 functionalities, but for some migration examples, we did not use any Rational tools. For those scenarios, we provide high-level steps for installing applications using the Administrative Console.

1. Once you are logged in the Administrative Console, go to **Applications** → **Install New Application**.
2. Specify the path of the file where your application is and click **Next**.
3. From this point on, you will go through several steps of configuration, depending on the deployment descriptors included in your application and the features needed. Follow the steps of the configuration, providing the necessary information to configure your application, then click **Finish** at the last step.
4. Click **Save** and when prompted, click **Save to the Master Configuration**.
5. Once you finish the installation of the application, the Administrative Console will show a list with the installed applications. You can start the new application by selecting it and clicking the **Start** button.



The operations described are just the first steps of the administration, and you will need to define more configuration settings in your applications. The configuration settings of the applications we migrated will be described as needed in the platform-specific chapters.

## Managing applications

The basic operations you can perform on applications are install, uninstall, start, stop, update, rollout update, remove file, export and export DDL. This options are available from **Applications** → **Enterprise Applications**.

The status of the applications can be any one of the following:

- ▶ Started
- ▶ Partial Start
- ▶ Stopped
- ▶ Partial Stop
- ▶ Unavailable
- ▶ Not applicable

To review and/or modify an application's configuration, just click the application name.

### 4.5.2 Universal Test Client

When you install Rational Application Developer V6, you will see that it contains a tool called Universal Test Client (UTC). This client is a Web application that you can use to test your EJBs.

To start the UTC, open a Web browser and type the following URL:

`http://localhost:9080/UTC`

The Universal Test Client will appear, as shown in Figure 4-7 on page 82.

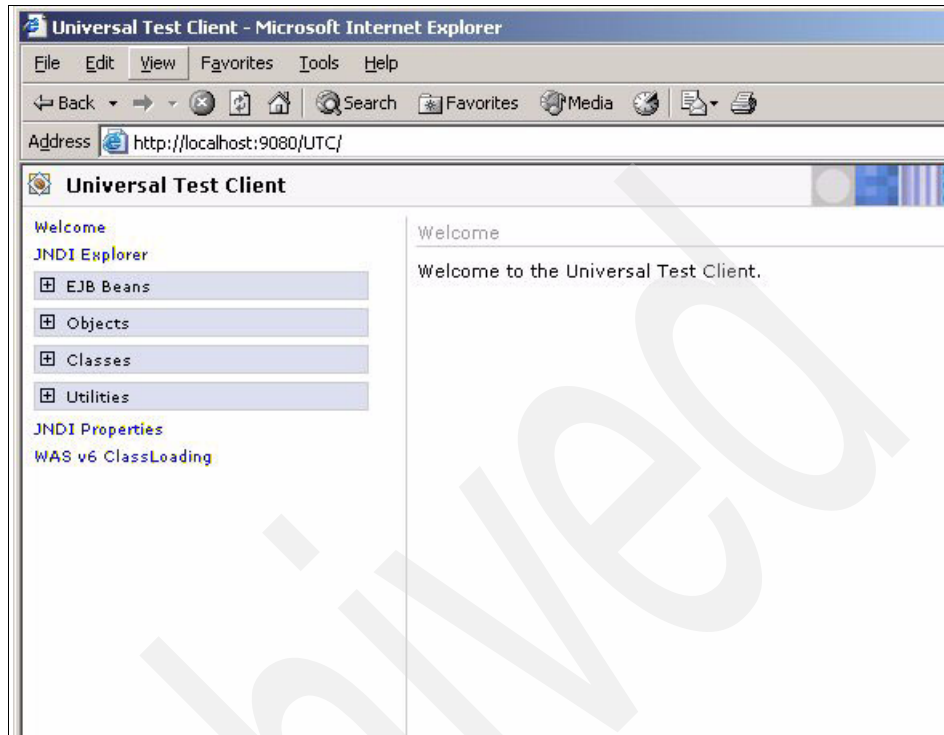


Figure 4-7 Universal Test Client welcome page

Click the **JNDI Explorer** link to view all the published resources and test them. At this point, running the UTC is just another way of verifying that your application server is installed and running properly.

You can also use the UTC directly from the Rational Application Developer V6 IDE to test EJB projects.

1. From the Project Explorer in the J2EE perspective, right-click the EJB Project you want to test inside of the **EJB Projects folder**, then select **Run** → **Run on Server**.
2. When the Server Selection dialog appears, select **WebSphere Application Server V6** and click **Finish**.
3. The server will be started and the EJB project will be deployed, if necessary.
4. When the Universal Test Client welcome page appears you can click **JNDI Explorer** and navigate it, looking for EJBs to test.

We used this tool during the migrations to test the correct migration of the EJB projects before starting with the migration of the Web projects. You can use this

tool during your migrations or your development process to perform a unit test of your EJBs.

## 4.6 Administrative scripting

WebSphere Application Server V6 supports the scripting of administrative tasks, including installation, configuration, deployment and runtime operations. The scripts are executed with the wsadmin scripting client and can be written in one of the following languages:

- ▶ JACL  
TCL-based scripting language
- ▶ Jython  
Python implemented in Java

The scripts have access to the following WebSphere Application Server V6 objects:

- ▶ AdminControl  
Contains commands for invoking and querying server objects.
- ▶ AdminConfig  
Allows you to manage the server configuration.
- ▶ AdminApp  
Used for managing applications.
- ▶ AdminTask  
Contains commands for running administrative commands, such as creating a cluster.
- ▶ Help  
Used for displaying help.

**Tip:** WebSphere Application Server V6 also contains Ant tasks that can be used to automate administrative tasks to run scripts.

For further information about Ant tasks, refer to the WebSphere Application Server V6 Information Center at the following URL:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.etools.j2eeapp.doc/topics/tjant.html>

WebSphere Application Server V6 includes a batch script for running Ant:

```
<was_home>\bin\ws_ant.bat
```

The script will include the WebSphere Application Server V6 classes.

The scripts are executed with the wsadmin tool. The wsadmin tool is located in:

```
<was_home>\profiles\default\bin
```

For further information and examples of scripting, refer to the WebSphere Application Server V6 Information Center at the following URL:

[http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/txml\\_script.html](http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/txml_script.html)

You can also download sample administration scripts from the following URL:

<http://www.ibm.com/developerworks/websphere/library/samples/SampleScripts.html>

## 4.6.1 Examples

In this section, we provide two basic examples of how to use the administrative scripting features in WebSphere Application Server V6.

**Note:** The scripts must be edited to match the names of the WebSphere Application Server V6 cell, node and server in your environment.

### Generating a thread dump

Thread dumps are useful when debugging, for example, thread locks and memory leaks.

The following script generates a thread dump for the WebSphere Application Server V6 instance server1.

#### *Example 4-2 JACL script for generating a thread dump*

---

```
set jvm [$AdminControl completeObjectName type=JVM,process=server1,*]  
$AdminControl invoke $jvm dumpThreads
```

---

Follow these instructions to run the script.

1. Save the script shown in Example 4-2 to:

```
<was_home>\profiles\<profile_name>\bin\threadDump.jacl
```

2. Execute the script with the following command:

```
wsadmin.bat -f threadDump.jacl
```

You should see the log message shown in Example 4-3.

#### *Example 4-3 Log message from the wsadmin tool*

---

```
WASX7209I: Connected to process "server1" on node 6690 using SOAP connector;  
The type of process is:  
UnManagedProcess
```

---

3. The tool generates a thread dump in the profile's root directory:

```
<was_home>\profiles\<profile_name>\javacore.20050602.042500.3808.txt
```

## **Starting the deployed application**

This example shows a more complete script that demonstrates how to start an application in WebSphere Application Server V6. It also demonstrates the use of arguments and other common scripting tasks.

Follow these instructions to run the script.

1. Save the script shown in Example 4-4 to:

```
<was_home>\profiles\<profile_name>\bin\startApp.jacl
```

2. Execute the script with the following command:

```
wsadmin -f startApplication.jacl "application_name"
```

#### *Example 4-4 JACL scripting example showing how to start an application*

---

```
proc startApplication {applicationName} {  
    global AdminControl  
    set cellName "Node01Cell"  
    set nodeName "Node01"  
    set processName "server1"  
    puts "Starting application '$applicationName'"  
    set applicationManager [$AdminControl queryNames  
cell=$cellName,node=$nodeName,type=ApplicationManager,process=$processName,*]  
    if {$applicationManager == ""} {  
        puts "Application manager not found"
```

```

        } else {
            $AdminControl invoke $applicationManager startApplication
$applicationName
        }
    }
    #=====
    #   Main method
    #=====
    if { !($argc == 1) } {
        puts "Please specify the application name"
    } else {
        set applicationName      [lindex $argv 0]
        startApplication $applicationName
    }
}

```

---

## 4.7 DB2 Universal Database V8.2

This section covers the steps we followed to install and configure DB2 Universal Database V8.2. For further DB2 installation and configuration instructions, refer to the DB2 Information Center at the following URL:

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp>

You can download DB2 Universal Database V8.2 Express Edition from the following URL:

<http://www.ibm.com/developerworks/downloads/im/udbexp>

Before you start, make sure that you will perform the installation with a user that belongs to the *Administrators* group, and with authority to create users and groups.

To install DB2 Universal Database V8.2, follow these steps.

1. Run the setup program.
2. In the setup launchpad window, select **Install Product** and then click **Next**.
3. In the licence agreement page, accept the license agreement and click **Next**.
4. When prompted to select an installation type, select **Compact installation** and click **Next**.
5. Select a drive and a directory to install and click **Next**. We installed DB2 Universal Database V8.2 in the following directory:  
C:\IBM\SQLLIB\
6. Next, you need to provide information for the administration server. Enter the user name and password that will be used by DB2 to log on and select the

option **Use the same user name and password for the remaining DB2 services.**

If you are using an existing user, make sure this user has the authority to perform the following actions:

- Act as part of the operating system
- Debug programs
- Create token object
- Increase quotas
- Lock pages in memory
- Log on as a service
- Replace a process level token

Otherwise, you can use a brand new user name and password and the installation program will create that user with all appropriate privileges. If the user that you insert exists, but does not have the privileges, it will be granted these privileges. In this case, you will need to log on with this account to make all those privileges effective.

Click **Next** to go to the following step.

7. A configuration window appears, allowing you to create DB2 instances. We clicked **Next** to accept the creation by default of a DB2 instance.
8. The installer will show you a summary of the products and features to be installed. Verify it and click **Install** to start the installation process.
9. A confirmation window will appear, informing you that DB2 was successfully installed. Click **Finish** to exit the setup.

## 4.8 Installation directories

The software we installed is sufficient to carry out all the migrations proposed in this redbook. As a summary, we present a list of the installed products and their directories.

- ▶ Rational Application Developer V6  
C:\IBM\Rational\SDP\6.0
- ▶ WebSphere Application Server V6 (Test enviroment)  
C:\IBM\Rational\SDP\6.0\runtimes\base\_v6
- ▶ WebSphere Application Server V6 (Second installation)  
C:\IBM\WebSphere\AppServer
- ▶ DB2 Universal Database V8.2  
C:\IBM\SQLLIB\





## Common migration issues

This chapter discusses some of the most common migration issues that we expect the reader to come across when migrating from other J2EE platforms to WebSphere Application Server V6.

Although the main idea behind Java and J2EE is portability and "Write Once, Run Anywhere" (WORA), this is not always the case, since the unique way in which each vendor implements the J2EE specification often leads to problems when migrating a J2EE application. Vendors also implement features that are not included in the J2EE specification. This is why Sun provides the J2EE Compatibility Test Suite (CTS), which consists of thousands of test cases based on the J2EE specification. The CTS test suite is used by Sun to verify that a specific J2EE application server complies with the specification. This ensures that applications deployed on one J2EE application server will also run on another vendor's application server.

J2EE applications can also conflict with the J2EE specification. The J2EE specification is so complex that it is practically impossible for developers to know all the details. Tools, such as Rational Application Developer V6 and the Java Application Verification Kit (AVK) for the Enterprise, can be used to verify that an application is written according to the specifications.

In this chapter, we provide a list of migration issues caused by the reasons mentioned above.

## 5.1 J2EE application server compability

Although the J2EE specification covers an increasingly wider spectrum of enterprise applications development, there is still the need for vendor-specific features.

If the vendor-specific features are used without proper planning, they will cause migration problems. Small differences and/or bugs in the vendor's implementation of the J2EE specification might also cause compatibility problems. We summarize these issues in the following sections:

- ▶ Differences in J2EE implementations
- ▶ Using vendor-specific features
- ▶ Class loader related problems
- ▶ Deployment descriptors

### 5.1.1 Differences in J2EE implementations

This section describes some of the problems we encountered during the migration process and which are caused by differences in how the J2EE specification is implemented by vendors. The J2EE specification leaves some room for interpretation, which leads to issues when migrating the application to a different application server.

#### **JBoss 3.2.7**

This is a summary of the problems we noticed when migrating from JBoss 3.2.7.

- ▶ Deployment descriptors fail validation in WebSphere Application Server V6  
JBoss 3.2.7 is not as strict as WebSphere Application Server V6 in validating J2EE applications. We noticed this while migrating the applications to WebSphere Application Server V6, which is much more strict in following the J2EE specification.
- ▶ Class loader problems  
The JBoss 3.2.7 class loader architecture is a major cause for problems when migrating to WebSphere Application Server V6. See 5.1.3, "Class loader related problems" on page 92 for more information.

#### **Apache Tomcat 5.5.9**

Apache Tomcat 5.5.9 follows the standards closely because it is the reference implementation for the JSP and Servlet specifications. However, we experienced the following problems:

- Use of single and double quotes in JSPs

WebSphere Application Server V6 does not use the same JSP compiler as Apache Tomcat 5.5.9. JSPs that work on Apache Tomcat 5.5.9 failed to run in WebSphere Application Server V6. In two of our migration exercises, we had to change single quotes to double quotes.

- JNDI resource references

When migrating xPetstore Servlet from Apache Tomcat 5.5.9 to WebSphere Application Server V6, we noticed that Apache Tomcat 5.5.9 allows invalid resource references in web.xml in the following format:

```
java:/comp/env/jdbc/xpetstoreDS
```

This JNDI resource reference does not work in WebSphere Application Server V6. The specification and DTD specify that the format must be:

```
java:comp/env/jdbc/xpetstoreDS
```

Note that there is no / preceding comp/.

## 5.1.2 Using vendor-specific features

When faced with the choice of using a vendor-specific feature, you have the option of not using it and accepting the limitations of the J2EE specification. This is good from a portability and migration point of view.

You can also decide to use the vendor-specific feature and accept the risk and potential future migration costs. If you decide to proceed with this option, ensure that you abstract away the implementation details, for example by designing interfaces, not concrete classes.

### BEA WebLogic Server 8.1

BEA WebLogic Server 8.1 contains many vendor-specific features. This list contains some of the more widely used.

- Startup and shutdown hooks

Startup and shutdown hooks are provided in BEA WebLogic Server 8.1 through the T3StartupDef and T3ShutdownDef interfaces. By using these, you can have a piece of your code invoked by BEA WebLogic Server 8.1 when the server is started and stopped.

When migrating to WebSphere Application Server V6, you can move the code to a servlet, configured to load on startup, to catch the startup event, and a ServletContextListener to catch the shutdown event. For further details about migrating WebLogic startup code to WebSphere, visit the following URL:

[http://www.ibm.com/developerworks/websphere/library/techarticles/0401\\_beaton/0401\\_beaton.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0401_beaton/0401_beaton.html)

► **Timer service**

BEA WebLogic Server 8.1 includes a service that can be used for triggering events at a specific time or interval.

Timers using this proprietary feature can be migrated to use the new Timer Service API in EJB 2.1 and WebSphere Application Server V6.

► **Automatic primary key generation for CMPs**

BEA WebLogic Server 8.1 contains two vendor-specific features for creating primary keys automatically:

- Native primary key generation
- Primary keys generated by a SEQUENCE table

WebSphere Application Server V6 does not have an equivalent feature that we can use. This means that you have to implement your own scheme for generating the primary key. The Trade sample application contains an example of this. It uses the KeySequenceDirect class for generating primary keys.

### **JBoss 3.2.7**

When migrating an application from JBoss 3.2.7, we experienced problems with EJBs that were using automatic primary key generation. As we just discussed for CMPs, WebSphere Application Server V6 does not support automatic primary key generation, so you will have to implement your own mechanism for generating the primary keys.

## **5.1.3 Class loader related problems**

The Java class loader is responsible for locating and loading classes. Application server class loaders are complex and so is the task of tracking down and solving these issues. To further complicate things, each J2EE application server uses a custom implementation of class loaders. This is because the J2EE specification does not currently cover class loaders. In this section, we provide some information about the class loader implementations and how they differ from the WebSphere Application Server V6 implementation.

Most class loader problems when migrating applications are related to the following:

► **Class loader implementation**

Each J2EE platform implements a class loader differently.

- **Shared libraries**

These are libraries that are made available to multiple applications or the whole server. These are located in different places and configured differently from server to server.

- **Bundled libraries**

Each platform is bundled with a unique set of frameworks and libraries, for example Apache Xerces and Apache Xalan. The versions of these libraries and the class loaders are different for each platform. Fixing these problems may require major changes to the source code and packaging.

### **JBoss 3.2.7**

We experienced difficulties in migrating an application that had been developed using JBoss 3.2.7. When deploying the application on WebSphere Application Server V6, the application threw `ClassNotFoundException` and other class loader exceptions. These errors are caused by the fact that JBoss 3.2.7 uses a custom class loader implementation, the Unified Classloader. The Unified Classloader is a repository of class loaders that all use the same namespace. This means a class is visible to all other classes. This design is radically different from the other application servers and creates problems when migrating to WebSphere Application Server V6.

WebSphere Application Server V6 uses a parent-child hierarchy which limits visibility.

To solve these problems, you might have to change the source code, the structure of the project and the way you package your application.

#### ***Bundled libraries***

JBoss 3.2.7 uses Log4j for its own logging. This means that applications built for JBoss do not have to include Log4j. WebSphere Application Server V6 does not include Log4j, so when we migrated the application to WebSphere Application Server V6, we received `ClassNotFoundException` exceptions that were related to Log4j. We fixed the problem by including Log4j in the EAR file.

### **Apache Tomcat 5.5.9**

We did not experience any class loader issues with Apache Tomcat 5.5.9, which uses a similar parent-child hierarchy class loader as WebSphere Application Server V6.

### **BEA WebLogic Server 8.1**

We did not experience any class loading issues migrating our sample applications from BEA WebLogic Server 8.1 to WebSphere Application Server

V6. BEA WebLogic Server 8.1 uses a similar parent-child class loader hierarchy as WebSphere Application Server V6.

## WebSphere Application Server V6

The WebSphere Application Server V6 class loader consists of four components:

- ▶ System classloader  
Provided by the Java virtual machine. Responsible for loading classes from the CLASSPATH, bootstrap and extensions class paths.
- ▶ WebSphere runtime class loaders  
Loads the WebSphere Application Server V6 runtime classes. All classes in the ws.ext.dirs class path are added to this class loader.
- ▶ Application class loaders  
Responsible for loading resources that are part of an EAR module, for example JAR, EJB and WAR files.
- ▶ Web module class loaders  
The Web module class loader loads the classes in WEB-INF/classes. It is possible to override this behavior and have the application class loader load the classes.

Each class loader is a child of the parent class loader, as illustrated in Figure 5-1 on page 95. When a resource needs to be loaded, all class loaders will by default ask the parent class loader to locate it. If it cannot be found, the resource is loaded by the child class loader. If none of the class loaders is able to find the resource, you will receive a `ClassNotFoundException`.

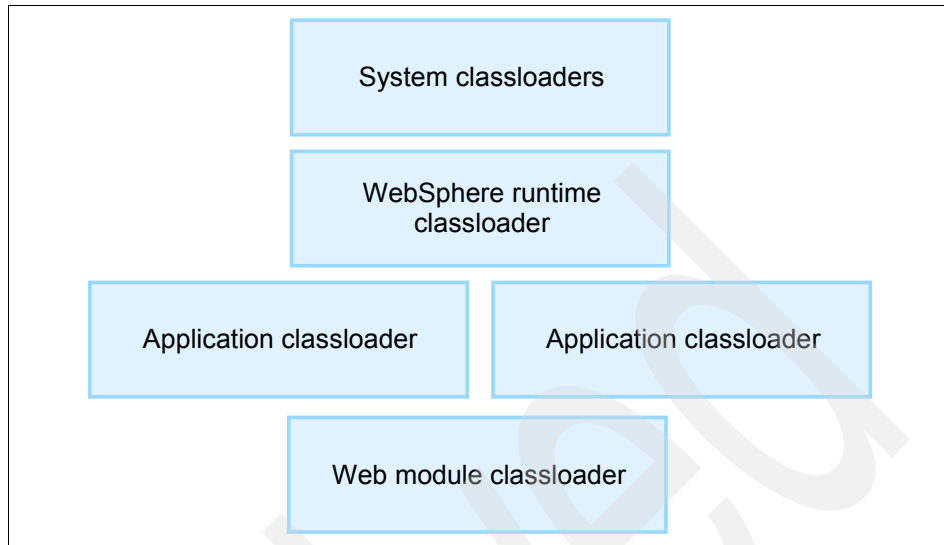


Figure 5-1 WebSphere Application Server V6 classloader in default configuration

### ***Class loader isolation policies***

The WebSphere Application Server V6 application and Web module class loaders can be configured to enable different packaging schemes. The default is *multiple*.

### ***Application class loader***

The application class loader can be configured to use the following modes.

- ▶ **Single**

When in single mode, the application class loader is shared by all modules in the EAR application. For example, all JAR, WAR and EJB modules have visibility to all classes located in all modules.

- ▶ **Multiple**

When in multiple mode, an application class loader is created for each module in the EAR application. This means that the different modules are isolated from each other.

### ***Web module class loader***

The Web module loads resources from the WEB-INF/lib and WEB-INF/classes directories in the WAR file. This is the default behavior and can be overridden by setting the policy to `Application`. This will make the application class loader load the resources from WEB-INF/lib and WEB-INF/classes, instead of the Web module class loader.

### ***Class loader modes***

WebSphere Application Server V6 supports two different class loader modes.

- ▶ **Parent first**

This is the default mode for all class loaders and it is the J2EE and Java standard. The loading of the resources is first delegated to the parent class loader. If the parent class loader is unable to find the resource, the child class loader tries to handle it.

- ▶ **Parent last**

Using the parent last class loading policy will make the class loader work opposite of the parent first mode. The resource is loaded from the application or Web module class loader first. If the resource is not found, the loading is delegated to the parent class loader.

### ***Shared libraries***

WebSphere Application Server V6 allows you to configure dependencies that are shared by either all applications in the server or just one specific application class loader. Shared libraries are configured using the admin console and can be specified at the application or server level.

### ***Bundled libraries***

WebSphere Application Server V6 contains some open source libraries that are located in the server classpath. If you are using the same libraries and there is a version mismatch, you might receive one of the following errors:

- ▶ `java.lang.NoSuchMethodError`
- ▶ `java.lang.ClassCastException`
- ▶ `java.lang.NoClassDefFoundError`

To fix these errors, you can try to experiment with different class loading policies, such as parent last instead of parent first.

Next, we list some of the most common libraries that are located in the WebSphere Application Server V6 server classpath.

- ▶ WebSphere Application Server V6 includes the Jakarta Commons Logging V1.2 (JCL), which is known to cause problems when migrating.

The following article describes how to solve JCL problems on WebSphere:

<http://www.ibm.com/support/docview.wss?uid=swg27004610>

- ▶ Another common error is caused by Apache Xalan and Xerces, which are usually bundled with application servers.
- ▶ JDom is also bundled with WebSphere Application Server V6.



### 5.1.4 Deployment descriptors

The J2EE specification does not cover all the aspects of application deployment and configuration. This is why we have vendor-specific deployment descriptors. In Table 5-1 on page 98, we have listed both the J2EE and vendor-specific deployment descriptors. Each application server uses a different set of files; this is because they are implemented differently and have different feature sets.

The biggest problems associated to deployment descriptors while migrating to WebSphere Application Server V6 can be summarized as follows.

- ▶ CMP to database schema mapping  
WebSphere Application Server V6 uses a completely different way of mapping CMPs to the database schema, compared to the other application servers covered in this book. The only way of creating these files is by using either Rational Application Developer V6 or the Java Application Verification Kit.
- ▶ The J2EE standard deployment descriptors are mapped to the WebSphere Application Server V6 specific deployment descriptors with IDs, as illustrated in Example 5-1 on page 98 and Example 5-2 on page 98. This means that the ID in the WebSphere Application Server V6 specific deployment descriptor must match the one in the J2EE standard deployment descriptor.
- ▶ WebSphere Application Server V6 validates the deployment descriptors more strictly than, for example, JBoss 3.2.7 and BEA WebLogic Server 8.1. This will lead to the following problems when migrating:
  - Deployment descriptors will fail validation because the elements are not in the proper order or the DTD is incorrect.
  - JSPs will not compile.

Next, we show an example of how the standard J2EE web.xml deployment descriptor and WebSphere Application Server V6 are mapped with IDs. In the example, a resource reference, having the ID ResourceRef\_2, is mapped to a JNDI name pointing to a resource that is managed by WebSphere Application Server V6.

*Example 5-1 Excerpt from web.xml*

```
...
    <resource-ref id="ResourceRef_2">
        <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
        <res-type>javax.jms.QueueConnectionFactory</res-type>
        <res-auth>Application</res-auth>
        <res-sharing-scope>Shareable</res-sharing-scope>
    </resource-ref>
...
```

*Example 5-2 Excerpt from WebSphere Application Server V6 specific*

```
...
    <resRefBindings xmi:id="ResourceRefBinding_1117409057536"
jndiName="jms/QueueConnectionFactory">
        <bindingResourceRef href="WEB-INF/web.xml#ResourceRef_2"/>
    </resRefBindings>
...
```

Table 5-1 lists the J2EE standard and vendor-specific deployment descriptors.

*Table 5-1 J2EE and vendor-specific deployment descriptors and configuration files*

Description	J2EE specification	WebSphere Application Server V6	BEA WebLogic Server 8.1	JBoss 3.2.7	Apache Tomcat 5.5.9
EAR application	application.xml	ibm-application-bnd.xmi ibm-application-ext.xmi	weblogic-application.xml	jboss-app.xml	N/A
WAR application	web.xml	ibm-web-bnd.xmi ibm-web-ext.xmi	weblogic.xml	jboss-web.xml	N/A
EJB application	ejb-jar.xml	ibm-ejb-jar-bnd.xmi ibm-ejb-jar-ext.xmi ibm-ejb-access-bean.xml	weblogic-ejb-jar.xml weblogic-cmp-rdbms-jar.xml	jboss.xml jaws.xml jbosscmp-jdbc.xml	N/A

Description	J2EE specification	WebSphere Application Server V6	BEA WebLogic Server 8.1	JBoss 3.2.7	Apache Tomcat 5.5.9
J2EE client application	application-client.xml	ibm-application-client-bnd.xmi ibm-application-client-ext.xmi	client-application.runtime.xml	jboss-client.xml	N/A
Server configuration	Not covered by the J2EE specification	Mainly located in <was_home>\profiles\<profile_name>\config\cells	Mainly located in <domain_home>\config.xml	Mainly located in the <jboss_home>\server directory	Mainly located in the <tomcat_home>\conf directory and <web_app>\META-INF/context.xml

## 5.2 Application portability

In this section, we describe some of the migration issues that affect application portability. Application portability, or the lack thereof, is perhaps the most important issue from a migration point of view.

### 5.2.1 Application packaging

Packaging is important from a portability and migration point of view. Each application is packaged differently. This, combined with the different class loader implementations in each application server, can potentially create problems when migrating. The application might run in the source environment but not in WebSphere Application Server V6. To avoid problems, always follow the J2EE specification, which contains specifications for application packaging.

An application is usually packaged into an EAR file which can consist of the following modules:

- ▶ EJB
  - Contains the EJB class files and deployment descriptors.
- ▶ WAR
  - Web application module containing deployment descriptors.

- ▶ JAR

A JAR file can contain common classes used by other modules and deployment descriptors for a J2EE application client.

- ▶ RAR

Resource Adapter archives is a JAR file containing resource adapters for the J2C architecture.

Figure 5-2 on page 101 illustrates the recommended way of packaging an application. The EAR package consists of the following modules.

- ▶ EJB module

The EJB module depends on Log4j, which is used for logging. It also depends on some utility classes. Each dependency is declared in the META-INF/MANIFEST.MF file.

- ▶ Web application module

The Web application module depends on Log4j, the utility library and the EJB module. Each dependency is declared in the META-INF/MANIFEST.MF file.

**Note:** The way you package the application may be different, depending on whether you are using Local or Remote EJB interfaces. If using Local EJB interfaces, as in our example, the client must have access to all EJB classes. If using Remote interfaces, you could optionally create a jar containing only the client view of the EJBs.

- ▶ Log4j.jar

Apache Log4j is a logging implementation used throughout the application. Third-party libraries that are used by the EJB module or more than one module should be placed in the application root.

- ▶ utility.jar

This is a utility library that contains common application code. Code that is used by more than one module should be located in this file.

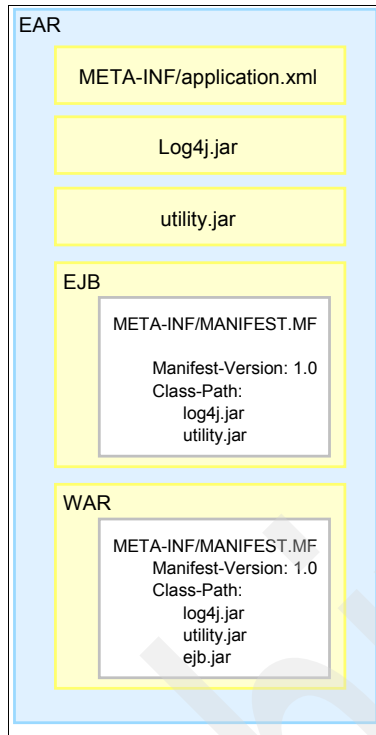


Figure 5-2 Recommended packaging structure of J2EE application

**Tip:** Always include the version number of libraries, frameworks and applications in either the MANIFEST.MF or the name of the file. During our migration process, we were unable to identify the version of many frameworks. Knowing the version number is critical when getting support.

## Sun Java Application Verification Kit

The Java Application Verification Kit (AVK) can help you when building portable applications by running a set of tests on the application package. These tests verify that the application is packaged according to the specification.

We used the Ant script shown in Example 5-3 to run the tests.

Example 5-3 Ant script used to run static tests in Java Application Verification Kit

```

<?xml version="1.0"?>
<project name="application verification" default="test" basedir=".">
  <property file="build.properties"/>
  <target name="test">

```

```

        <echo message="${dir.avk}"/>
        <taskdef name="ArchiveTest"
classname="org.apache.tools.ant.taskdefs.optional.sun.verifikation.StaticArchiv
eTest">
        <classpath>
        <pathelement path="${classpath}"/>
        <fileset dir="${avk.home}/lib/">
            <include name="javke-ant.jar"/>
        </fileset>
        </classpath>
        </taskdef>
        <ArchiveTest appName="${file.archive}" reportingOpts="w" />
    </target>
</project>

```

---

The Ant script stores its configuration in a property file shown in Example 5-4.

*Example 5-4 Property file for the static test*

---

```

# Path to the AVK installation directory. Required by the AVK ant task
avk.home=C:/java/javke1.4.1/
# The path to the package we want to test
file.archive=./sampleApp.ear

```

---

The Java Application Verification Kit static tests are run by issuing the **ant** command from the command line. This will run a series of static tests on the application and create a report.

```
<avk_home>\reporttool\static\verifierSummary.html
```

The report will contain errors and warnings as shown in Figure 5-3 on page 103.

Java AVK for the Enterprise - Static Archive Test Summary	
Static Archive Results Summary	
Failures:	12
Warnings:	23
EJB Module	
Failures:	<a href="#">Failed</a>
Warnings:	<a href="#">click to see Warnings</a>
Web Module	
Failures:	<a href="#">Failed</a>
Warnings:	<a href="#">click to see Warnings</a>
Execution Errors	
Errors:	No Errors

Figure 5-3 AVK report

Clicking the EJB module link will show you all errors. Example 5-5 shows you an error that is related to packaging. The error is caused by the EJB module, which is referencing a library that is not in the application.

*Example 5-5 Packaging error detected by AVK*

---

```
For [ example#ejb-app.jar#ReportHandlerBean ] classes [ org.apache.log4j.Level
org.apache.log4j.Logger org.apache.log4j.PatternLayout
org.apache.log4j.ConsoleAppender ] referenced by [
example.ReportHandlerBean\util.Logger] are not found
```

---

The static tests in Java Application Verification Kit will help you assess the problems with migrating an application.

## Migration problems

During the migration of the sample applications, we found the following packaging-related problems.

- ▶ EJB module references classes in the WAR module

The application worked on JBoss 3.2.7 because of the JBoss Unified Classloader flat hierarchy. However, it stops working when deployed to WebSphere Application Server V6, because the EJB cannot view what the WAR module contains.

**Tip:** The EJB module should never depend on classes in the Web module. This creates a dependency which does not work on platforms other than JBoss 3.2.7. Instead, you should extract these classes from the Web application module and put them in a utility JAR. Use the META-INF/MANIFEST.MF file to declare the dependency.

- ▶ EJB module references libraries located in the WAR module

In one application, EJBs imported classes from both Apache Axis and Apache Struts. The required jar files were located in the Web modules WEB-INF/lib folder and only visible to that module.

**Tip:** Dependencies on libraries that are shared between J2EE modules in an application should be in the root of the EAR file. The modules that have the dependencies should declare the dependency by using the META-INF/MANIFEST.MF file.

## 5.2.2 Use of native code

Java allows the use of native code, for example written in C or C++, with the use of Java Native Interface (JNI). Using native code can potentially cause problems when migrating to a different operating system.

One of the sample applications that we migrated uses a Windows DLL to communicate with a mail server. It is unclear why the application uses the Windows DLL, instead of using the JavaMail API, which is part of the J2EE specification.

## 5.2.3 Database-related issues

In our migration examples, we used applications that were developed for and running on different databases, including HSQL and MySQL. We migrated these to use DB2 Universal Database V8.2. In this section, we describe the issues that we encountered and the solutions to them:



► Migrating database schemas

Most of the issues we had with migrating database schemas were easy to solve either by manually editing the DDL or SQL scripts or running them through a tool. For issues related to MySQL to DB2 migration, there is a redbook available:

<http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg247093.html>

The more difficult problems were the following.

– Data type overflow

When moving one database to DB2 from MySQL, we had to change the data type of one column from DATE to TIMESTAMP because the application was inserting TIMESTAMP.

– Automatic primary key generation

MySQL supports AUTO\_INCREMENT columns. The DB2 equivalent for this is the *identity* column. We had to change the schema to use identity columns.

► Mapping database schemas to CMPs

In our migration examples, we had to map CMPs to the existing database schemas. This is a complex task and requires the use of either Rational Application Developer V6 or Java Application Verification Kit. In theory, it is possible to manually edit XML and XMI deployment descriptors, but we do not recommend trying this.

The database mapping is done with the *EJB to RDB mapping tool*, which supports the following options:

– Top down

This will create a database schema from existing CMPs.

– Bottom up

By using the bottom up approach, you can generate CMPs from an existing database schema.

– Meet-in-the-middle

This option is probably the one that you will want to use. It allows you to automatically map the CMPs to database tables and columns. The tool will try to match column and table names to the CMP. If the column or table names do not match, you have the option of creating the mapping manually using the Mapping Editor.

- ▶ Primary key generation

Both JBoss 3.2.7 and BEA WebLogic Server 8.1 contain vendor-specific EJB extensions that provide support for automatic primary key generation, as we described in 5.1.2, “Using vendor-specific features” on page 91.

## 5.2.4 JMS

The JMS messaging configuration in WebSphere Application Server V6 is different from that of the other application servers we cover in this book. It is also different from that of WebSphere Application Server V5. The major differences are summarized in the following list.

- ▶ You cannot use the JMS default messaging engine provided by WebSphere Application Server V6 with listener ports.

We have two options to solve this problem:

- Use the JMS Activation Specification instead of listener ports.
- Use IBM WebSphere MQ, or other supported JMS server instead of the default messaging engine.

- ▶ JMS messaging configuration.

Before being able to configure JMS connection factories, queues and topics, you have to configure the following components:

- Service integration bus
- Bus member
- Bus destinations for the JMS queues and topics
- JMS activation specification for JMS queues and topics

The steps required for configuring JMS messaging are described in detail in the migration examples on each of the platform-specific chapters, as needed.

## 5.2.5 JNDI

There are some differences in WebSphere Application Server V6 related to JNDI that might affect migration projects:

- ▶ InitialContext factory

The InitialContext factory for WebSphere Application Server V6 is:

```
com.ibm.websphere.naming.WsnInitialContextFactory
```

In Example 5-7 on page 107, we show how to look up a datasource through JNDI.

- Direct JDBC datasource lookups are deprecated

If you use the JNDI name to look up a JDBC datasource, you will get the warning shown in Example 5-6. Instead of looking up the JNDI name, we have to use a resource reference defined in the web.xml file.

Example 5-6 shows the warning message that WebSphere Application Server V6 brings up when performing a direct lookup of a JDBC datasource through its JNDI name.

**Restriction:** The JNDI lookups will only work with the IBM JDK.

*Example 5-6 Warning message*

---

00000042 ConnectionFac W J2CA0294W: Deprecated usage of direct JNDI lookup of resource jdbc/xpetstoreDS. The following default values are used:  
[Resource-ref settings]

```
res-auth:                1 (APPLICATION)
res-isolation-level:      0 (TRANSACTION_NONE)
res-sharing-scope:        true (SHAREABLE)
loginConfigurationName:   null
loginConfigProperties:     null
[Other attributes]
```

```
res-resolution-control:   999 (undefined)
isCMP1_x:                  false (not CMP1.x)
isJMS:                      false (not JMS)
```

---

Example 5-7 demonstrates the correct way of finding a JDBC datasource through a JNDI lookup. The JSP will only work on WebSphere Application Server V6, since it relies on vendor-specific classes.

**Note:** The default IIOP port is 2809. It might be different in your environment.

*Example 5-7 Looking up a JDBC datasource through JNDI and a resource reference*

---

```
<%@page import="javax.naming.*,java.util.Hashtable"%>
<%
    String dataSource = "java:comp/env/jdbc/xpetstoreDS";
%>
Looking up data source <%= dataSource %><br/>
<%
    try {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.ibm.websphere.naming.WsnInitialContextFactory");
```

```

env.put(Context.PROVIDER_URL, "iiop://localhost:2809");

Context initialContext = new InitialContext(env);
javax.sql.DataSource ds = (javax.sql.DataSource) initialContext
    .lookup(dataSource);
%>
Success
<%
    } catch (Exception e) {
        out.write(e.getMessage());
        e.printStackTrace();
    }
%>

```

---

## 5.2.6 J2EE application clients

A J2EE application client is a standalone client running in a separate process from the application server. The client is given access to resources in the WebSphere Application Server V6 J2EE Client Container. The J2EE application client can be used to communicate with EJBs or other resources deployed on the application server. There are several use cases where J2EE application clients are appropriate:

- ▶ Alternative user interface for the application
- ▶ Running unit tests
- ▶ Administrative tasks

Migrating J2EE application clients is not covered in practice by this book. We only provide an overview of the options provided by WebSphere Application Server V6.

WebSphere Application Server V6 J2EE application clients are executed with the following command:

```
<was_home>/bin/launchClient.bat client.ear
```

The script will set up all necessary classpath and configuration variables and run the J2EE application client, which must be packaged as an EAR file.

### Thin clients

Thin application clients provide a more lightweight alternative to J2EE application clients. The thin clients access WebSphere through RMI-IIOP and do not have access to all the services provided by WebSphere.

For example, JNDI lookups do not have access to `java:comp/env`; instead, the thin client must provide the fully qualified path, including the physical location, as illustrated in Example 5-8.

*Example 5-8 Thin client looking up an EJB deployed on a single server*

---

```
...
String path = "cell/nodes/nodeName/servers/serverName/relativeJndiName";
Object object = initialContext.lookup(path);
EJBHome ejbHome = (EJBHome) javax.rmi.PortableRemoteObject.narrow(object,
EJBHome.class);
...
```

---

To run thin clients, you need to install the *J2EE and Java thin application client*, which is part of the WebSphere Application Server V6 installation package and can be installed separately.

Refer to the WebSphere Application Server V6 Information Center for more information about how to run and develop thin clients:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.eexpress.doc/info/exp/ae/tcli\\_developthin.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.eexpress.doc/info/exp/ae/tcli_developthin.html)

## 5.3 J2EE 1.3 to 1.4 migration considerations

J2EE is evolving all the time, and new specifications and requirements are defined periodically with a new J2EE specification. WebSphere Application Server V6 supports all of the J2EE 1.4 related technologies, but you can still deploy J2EE 1.3 applications as well.

If you plan to maintain your applications, you should consider migrating them to J2EE 1.4. This way, you will be up to date, and the changes in the next J2EE specifications will be easier to carry out.

While the main changes of J2EE 1.4 are related to Web services, the introduction of the JAX-RPC and SAAJ APIs and the new messaging bus architecture supported by MDBs means that the remaining technologies and APIs are not changing as much, and in some cases, there are only additions or clarifications of ambiguities in the previous version.

Next, we give an overview of the changes and new functionalities in the main technologies of J2EE 1.4.

### 5.3.1 Enterprise Java Beans

EJB 2.1 adds new functions to the EJB QL language, a timer service API to schedule tasks, and enhanced Message Driven Beans which can support any messaging system, not just JMS. EJB 2.1 also includes better support for Web services, through the JAX-RPC and JAXM APIs.

### 5.3.2 Java Server Pages

The new JSP 2.0 comes with the introduction of the JSP Standard Type Library (JSTL) specification, an expression language and some clarifications about ambiguities in the previous version. This last item is very important for migration projects, because the ambiguities are caused by differences in the J2EE implementations of the different vendors.

### 5.3.3 Servlets

There are no major changes in the Servlet 2.4 specification. Some clarifications about the use of filters, the possibility of including servlets as welcome pages, and the addition of two new servlet request events (`ServletRequestListener` and `ServletRequestAttributeListener`) are some of the changes in this version. It is unlikely that these changes could become migration issues.

For a complete list of new features in J2EE 1.4 refer to:

<http://www.jcp.org/en/jsr/detail?id=151>

For deprecated functions, refer to:

<http://java.sun.com/j2ee/1.4/docs/api/deprecated-list.html>

## Migrating from BEA WebLogic

This chapter describes the migration process and individual migration issues and solutions we found while migrating two J2EE 1.3 applications from the latest available version of BEA WebLogic Server 8.1 to WebSphere Application Server V6. We provide step-by-step instructions for migrating these applications.

The compatibility of application servers these days is surprisingly good. We did not have major problems with migrating our sample applications. In both cases, we were able to deploy WebLogic compiled EAR files into WebSphere without having to change Java code. The scope of change was limited to deployment descriptors. This is as expected since the J2EE specification allows for vendor-specific deployment descriptors.

## 6.1 Introduction

BEA WebLogic Server 8.1 was released in May of 2003 and at the time of writing this book, it is the latest available version of the J2EE runtime from BEA. Soon after this book is published, BEA is expected to release WebLogic Server 9.0 for which a beta version is available today. WebLogic Server has had a long history since it started as a Tengah product in the late 90s.

During our migration exercise for this platform, we used two different J2EE 1.3 sample applications.

The Trade 3.1 application that we have selected for migration was developed by IBM for performance testing purposes and does not use any non-J2EE features of BEA WebLogic Server 8.1. We have selected Trade because it is very well built application, uses most of the J2EE 1.3 APIs as shown on Table 1-1 on page 3, implements a number of good programming techniques for performance and implements a number of patterns that you may want to reuse in your application development efforts. However, there is a possibility that your application may have some dependency on WebLogic proprietary extensions, or even worse, it may have been constructed using the BEA WebLogic Workshop development tool. If this is the case, your application is likely not portable to other application servers and requires a redesign and rewrite.

The second application we have migrated is xPetstore EJB 3.1.3, which is a rewrite of Sun's Petstore application. This may not be the typical application that you find in the enterprise since it was designed and built to be portable across multiple application server platforms. The reason we have selected this application is that it was not designed for WebSphere and there is no evidence that anyone had ever tried it in WebSphere with DB2. The application uses a number of elements of J2EE 1.3 as shown in Table 1-1 on page 3.

Perhaps this is a good time to mention Sun's Java Application Verification Kit (AVK) for the Enterprise as a tool intended to help developers test their applications for the correct use of J2EE APIs and portability across J2EE compatible application servers, and to help developers avoid inadvertently writing non-portable code. See more details at the following URL:

[http://java.sun.com/j2ee/verified/avk\\_enterprise.html](http://java.sun.com/j2ee/verified/avk_enterprise.html)

## 6.2 Prerequisites and assumptions

There are some prerequisites and assumptions we need to consider before carrying on with the migration examples. One basic requirement is that the reader should have the "destination" environment installed and configured as



described in Chapter 4, “Installation and configuration” on page 63. This should provide some initial background knowledge if the reader is not yet familiar with IBM products.

The reader should also be familiar with the J2EE specification and architecture, be able to understand and run basic SQL commands. The reader should also be familiar Rational Application Developer V6 and Ant. Last but not least, the reader should be knowledgeable about BEA WebLogic Server 8.1.

The following software should be installed before starting the migration:

► J2SE 1.4

We use J2SE 1.4 since it is supported by WebSphere Application Server V6 and we need to make sure our applications work well in this JVM. The WebSphere Application Server V6 installation includes JDK 1.4.2 and you do not need to install additional JDKs on your machine.

► Ant 1.5.1

Apache Ant is a Java-based build tool. It is similar to the Make tool, but has many enhanced features that make it a very good choice for Java applications. This is required for migrating xPetstore EJB application.

► IBM WebSphere Application Server V6

Before you start migrating to WebSphere Application Server V6, we recommend that you play with sample applications shipped with the product and also read publications listed in “Related publications” on page 325. Instructions on how to install and configure WebSphere Application Server V6 are covered in Chapter 4, “Installation and configuration” on page 63.

► IBM DB2 UDB 8.2

WebSphere Application Server V6 supports many databases, but we have decided to use DB2 for all our applications. Instructions on how to install and configure DB2 Universal Database V8.2 can be found in Chapter 4, “Installation and configuration” on page 63.

► IBM Rational Application Developer V6

We used the J2EE Competitive Migrator plugin for Rational Application Developer V6 that supports the migration of J2EE applications developed for competitive application servers to WebSphere Application Server. Currently the tool supports the migration of EJB modules deployed for BEA WebLogic Server to WebSphere Application Server version 5.0, 5.1, or 6.0. Instructions on how to install and configure Rational Application Developer V6 can be found in Chapter 4, “Installation and configuration” on page 63.

## 6.3 BEA WebLogic Server 8.1 installation

This section provides very high-level steps and tasks for installing and configuring BEA WebLogic Server 8.1 SP4 on the Microsoft Windows 2000 platform. Instructions on how to install and configure the WebSphere destination environment are covered in detail in Chapter 4, “Installation and configuration” on page 63.

Detailed instructions for installing, configuring and managing BEA WebLogic Server 8.1 are provided in the product documentation guides available at the following URL:

<http://e-docs.bea.com>

The following list highlights the high-level tasks we performed to install and configure our initial environment in order to have a clean setup as our starting point for deploying the two sample applications.

1. Download BEA WebLogic Platform 8.1, which is a suite that includes WebLogic Server 8.1 SP4 and other BEA products from the following URL:

<http://commerce.bea.com>

2. Install the software in the following directory (later in the text referred to as <weblogic\_home>):

D:\bea

3. Launch the BEA WebLogic Configuration Wizard by clicking **Start** → **Programs** → **BEA WebLogic Platform 8.1** → **Configuration Wizard**.
4. From the BEA WebLogic Configuration Wizard, select **Create a new WebLogic configuration**.
5. Select the **Basic WebLogic Server Domain** template.
6. When prompted for Express or Custom configuration, select **Express**.
7. When prompted for Configure Administrative Username and Password, accept the default username and use the same value for the password; for both cases, we used weblogic.
8. For the WebLogic Configuration Startup Mode, use the default offered, **Development Mode**.
9. For the Java SDK Selection, specify JRockit SDK 1.4.2 from the BEA Supplied SDKs list.
10. Specify the Configuration Name TradeDomain for migrating the Trade application.
11. For the Configuration Location, specify the following directory:

D:\bea\user\_projects\domains\TradeDomain

12. Run the configuration wizard a second time and create a second domain for the xPetstore EJB application. This time, create *xpetstore-cmp* in the following directory:

D:\bea\user\_projects\domains\xpetstore-cmp

## 6.4 Trade for BEA WebLogic Server 8.1 migration

The Trade 3.1 application was developed by IBM to evaluate J2EE performance issues. Although at this time, we are not interested in any performance evaluation, this application provides a wide range of features that we were interested in covering during the migration exercise. The latest version of Trade is V6 and it is built as a complete J2EE 1.4 application that utilizes most components of the specification, but due to the fact that BEA WebLogic Server 8.1 is not J2EE 1.4 certified, we had to use Trade 3.1. Both versions of Trade for WebSphere are available for download from the following URL:

<http://www.ibm.com/software/webservers/appserv/was/performance.html>

The Trade 3.1 version we used for BEA WebLogic Server 8.1 is a slightly modified version from the one available for download for WebSphere Application Server; refer to Appendix B, “Additional material” on page 323 for details on how to download the Trade 3.1 we used for BEA WebLogic Server 8.1.

The modifications made to the application are in terms of configuration files and deployment descriptors. The core of the application remains unchanged, therefore during this migration we do not expect any issues with the Java code and will mainly focus on migrating deployment descriptors and creating resources in WebSphere Application Server V6 to support the application.

Some of the main features we wanted to cover and were provided by this application make extensive use of J2EE 1.3 APIs as shown on Figure 6-1 on page 116:

- ▶ JMS 1.0.2
- ▶ JDBC 2.1
- ▶ Entity and Stateless Session EJB 2.0
- ▶ Primary Key generation pattern for CMP EJB 2.0
- ▶ Message Driven Beans 2.0
- ▶ 2-phase commit transactions
- ▶ JSP 1.2
- ▶ Servlet 2.3

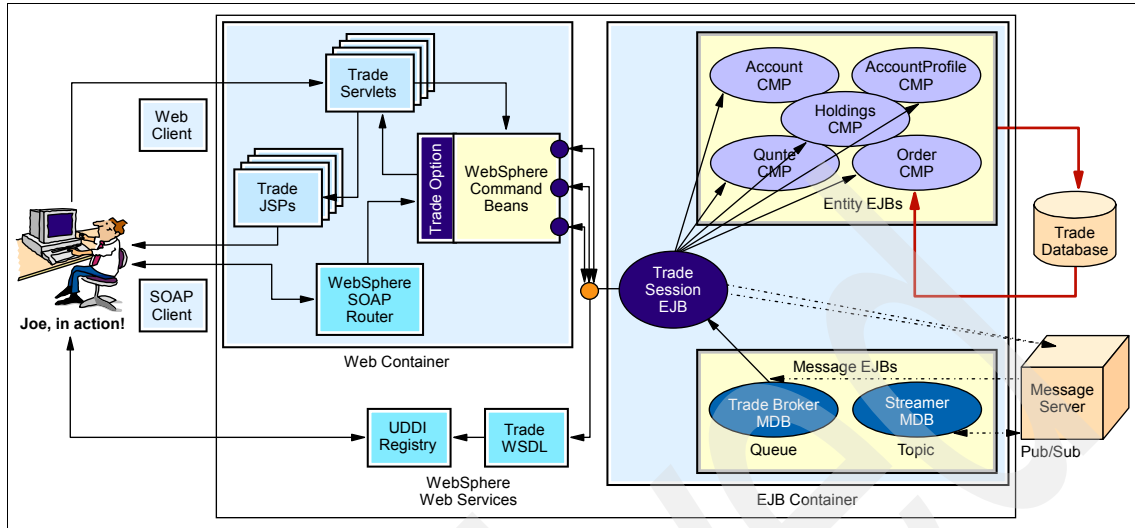


Figure 6-1 Trade3 J2EE components

### 6.4.1 Migration approach

We decided to perform the Trade migration as explained in 3.4.4, “Alternative 4: Developing using Rational Application Developer V6” on page 56.

In addition to Rational Application Developer V6, we will use the J2EE Competitive Migrator Plugin 1.2.0 for Rational Application Developer V6. We decided to use the plugin because it supports the migration of EJB modules from BEA WebLogic Server 8.1 to WebSphere Application Server V6.

The migration is performed in the following steps:

1. Verify that application works in its original environment. In this case, we will deploy and run the Trade 3.1 application in BEA WebLogic Server 8.1.
2. Import EAR file into Rational Application Developer V6.
3. Analyze and fix problems reported.
4. Migrate the EJB deployment descriptors using the J2EE Competitive Migrator Plugin 1.2.0 plugin.
5. Build Trade for WebSphere Application Server V6 using Rational Application Developer V6.
6. Deploy the application on WebSphere Application Server V6 and identify and solve all problems reported.

## 6.4.2 Configuring the initial environment

After installing BEA WebLogic Server 8.1 and creating a new server configuration, the next step is to deploy the sample application to our initial environment. This section describes the procedures we followed for creating a sample database, configuring BEA WebLogic Server 8.1 to hold the Trade application, deploying the application and testing.

### Getting the Trade application

Refer to Appendix B, “Additional material” on page 323 for details on how to download the Trade 3.1 application for BEA WebLogic Server 8.1

Once you have downloaded the application, create a Trade3 directory and unzip the downloaded file with the name trade3-WebLogic.zip. The zip contains the following files and directories:

- ▶ db2\_schema.ddl  
Script for creating the database in DB2 Universal Database V8.2.
- ▶ trade3wls.ear  
Trade application binaries.
- ▶ trade3EJB  
Directory containing source code for the EJB module.
- ▶ trade3WSAppClient  
Directory containing the source code for a standalone client that uses Web services to communicate with Trade. The source code is only needed at compile time. The compiled classes are not included in the EAR file and are not a runtime dependency, which we notice later when starting the migration.
- ▶ trade3AppClient  
Directory containing source code for the J2EE client application for Trade.
- ▶ trade3Web  
Directory containing the source code for the Web module.

In our scenario, we used the following directory to unzip the application (later referred to as <source\_home>):

D:\sampleApp\Trade3

### Creating the sample application database

In this section, we describe the necessary steps for creating the Trade database. The database schema is included in the installation package.

1. From a command line, you need to start DB2 CLP (command line processor) as follows:  

```
cd <source_home>
db2cmd
```
2. Now, while you are in DB2 command prompt, execute the following commands to create the Trade database and tables.

**Note:** The user you use to connect to the database will define the name of the schema where the DDL will create the tables.

```
db2 create db trade3db
db2 connect to trade3db user db2admin using its0ra10
db2 -tvf db2_schema.ddl
db2 disconnect all
db2 update db config for trade3db using logfilsiz 1000
db2 update db cfg for trade3db using maxappls 100
db2stop
db2start
db2 connect to trade3db user db2admin using its0ra10
cd <db2_home>\bnd
db2 bind @db2cli.lst blocking all grant public
```

This concludes the creation of the database and tables; at this point, the database is empty and ready to get the data loaded. The creation of initial data in the database will be done by the application when it is deployed.

### Adding the DB2 JDBC driver to the WebLogic classpath

As we mentioned in 6.3, “BEA WebLogic Server 8.1 installation” on page 114, we created a new WebLogic configuration to have a clean environment to start deploying and configuring the Trade application. The creation of this new configuration is an optional step. You can use any of your existing WebLogic domains for this task.

1. For the new WebLogic configuration, update the startWebLogic.cmd file by adding the location of the DB2 JDBC drivers to the end of the classpath. Make sure that you specify both db2java.zip and db2jcc.jar files as shown in Example 6-1.

*Example 6-1 Excerpt from the startWebLogic.cmd configuration file*

---

```
. . .
set
CLASSPATH=...<_standard_weblogic_classpath_>...;%CLASSPATH%;<db2_home>java\db2j
ava.zip;<db2_home>java\db2jcc.jar
. . .
```

---

In our scenario, the DB2 JDBC drivers are located in the following directory:

C:\IBM\SQLLIB\java

2. We start the BEA WebLogic Server 8.1 instance from the command line:

```
d:
cd \bea\user_projects\domains\xpetstore-cmp
startWebLogic.cmd
```

3. Once the server is up and running, access the Console Login. When we created the new WebLogic configuration, we accepted the default values for port and location. In our scenario the Console Login can be accessed at the following URL:

<http://localhost:7001/console>

The username and password required for login were specified during the creation of the new WebLogic configuration. In our scenario, we used `weblogic` for both the username and password.

## Creating a new connection pool and the datasource

1. Once within the WebLogic Server Console, navigate to **TradeDomain** → **Services** → **JDBC** → **ConnectionPool** and create a new connection pool with the following characteristics.

Table 6-1 Connection pool values

Parameter	Value
Database type	DB2
Database Driver	IBM's DB2 Driver (Type 2 XA) Versions:7.X, 8.X
Name	Trade3 Connection Pool
Database Name	trade3db
Database User Name	db2admin
Password	its0ral0

2. Test the driver configuration and then create and deploy the connection pool to the only server available in this new WebLogic configuration.
3. Once the connection pool has been created and successfully tested, proceed to create a datasource with the following characteristics.

Table 6-2 Datasource values

Parameter	Value
Name	Trade JDBC Data Source
JNDI Name	jdbc/TradeDataSource
Pool Name	Trade3 Connection Pool
Honor Global Transactions	Yes
Emulate Two-Phase Commit for non-XA Driver	No

4. Make sure that the server "myserver" is selected for target the datasource and then click **Create** to finalize the datasource creation.
5. Once again, go to the connection pools and select the connection pool you just created (Trade3 Connection Pool). In Configuration/Connections, under Advanced Options, mark the three checkboxes for **Test Reserved Connections**, **Test Created Connections** and **Test Released Connections** respectively.
6. In the Test Table Name field, replace the content with the following:  
DB2ADMIN.ACCOUNTEJB
7. Apply the changes and from the Testing tab, test the JDBC connection by clicking **Test Pool**. You should receive a successful connection message on "myserver".

## Creating JMS resources

1. From the WebLogic Server Console home page, create a new JMS Server with the following characteristics:

Table 6-3 JMS Server values

Parameter	Value
Name	TradeJMSServer
Target	myserver

2. From the Configuration tab, still within the TradeJMSServer, configure the destinations by creating a new JMS Queue and an new JMS Topic.  
Create a new JMS Queue with the following characteristics:



Table 6-4 JMS Queue values

Parameter	Value
Name	jms/TradeBrokerQueue
JNDI Name	jms/TradeBrokerQueue

From the JMS Destinations, create a JMS Topic with the following characteristics:

Table 6-5 JMS Topic values

Parameter	Value
Name	jms/TradeStreamerTopic
JNDI Name	jms/TradeStreamerTopic

Create a second JMS Topic with the following characteristics:

Table 6-6 JMS Topic values

Parameter	Value
Name	jms/TradeStatsTopic
JNDI Name	jms/TradeStatsTopic

- Next, you need to create two connection factories; select **TradeDomain** → **Services** → **JMS** → **Connection Factories** and create the connection factories with the following characteristics:

Table 6-7 Queue connection factory values

Parameter	Value
Name	jms/QueueConnectionFactory
JNDI Name	jms/QueueConnectionFactory
Target	myserver
Transactions	enable XA Connection Factory

Table 6-8 Topic connection factory values

Parameter	Value
Name	jms/TopicConnectionFactory
JNDI Name	jms/TopicConnectionFactory

Parameter	Value
Target	myserver
Transactions	enable XA Connection Factory

## Deploying the sample application and populating the database

The next step is to deploy the Trade application. To do this, you need to upload the Trade3WLS.ear file by selecting **Deploy a new Application...** from the Server Console home page and then selecting **Upload your file(s)**.

Click **Browse** to navigate and select the **Trade3WLS.ear** file. After clicking **Upload**, make sure you select the radio button next to **trade3wls.ear**, click **Continue** and then **Deploy**. This process is very quick and normally completes in less than a minute. At this point, you should see two successful messages, one for the deployment status for the EJB Modules and another for the deployment status for the Web Application Modules.

Test the newly deployed application by accessing the following URL from your Web browser:

<http://localhost:7001/trade>

From the Trade home page, select **Configuration** and then select **(Re)-populate Trade Database** to start loading the database with sample data. This process takes a few minutes depending on the hardware configuration used. When finished, you should see 1000 quotes created and 500 users registered. On the left pane, click **Go Trade!** and browse the application.

Trade also has another interesting feature. If you click the **Configuration** link on your left-hand side and then click **Trade Scenario**, you can see how the application automatically “navigates” itself simply by clicking the **Reload** button in your browser as shown in Figure 6-2 on page 123.

This feature randomly selects the next action to be performed and populates all the data instead of the user having to do it by hand. This greatly simplifies performance scripts. If you want to performance test this application, you do not need to navigate all windows and enter all the data. You can simply point your stress tool to the URL <http://localhost:7001/trade/scenario> and configure your stress test tool to see how many hits and how many concurrent users will access this URL, and then start your test.

Trade Account										Trade3
<a href="#">Home</a>	<a href="#">Account</a>	<a href="#">Portfolio</a>	<a href="#">Quotes/Trade</a>	<a href="#">Logout</a>	<a href="#">operation schematic</a>					

Thu Jun 02 23:40:36 EDT 2005

### Account Information

[account created:](#) 2005-05-24 17:01:13.336      [last login:](#) 2005-06-02 23:40:29.64  
[account ID:](#) 255      [total logins:](#) 1      [cash balance:](#) 218273.20  
[user ID:](#) uid:255      [total logouts:](#) 0      [opening balance:](#) 245909.00

**Total Orders: 2**      [show all orders](#)

<a href="#">order ID</a>	<a href="#">order Status</a>	<a href="#">creation date</a>	<a href="#">completion date</a>	<a href="#">txn fee</a>	<a href="#">type</a>	<a href="#">symbol</a>	<a href="#">quantity</a>	<a href="#">price</a>	<a href="#">total</a>
1251	closed	2005-05-24 17:01:13.346	2005-05-24 17:01:13.346	24.95	buy	s:686	150.0	66.69	10003.50
1252	closed	2005-05-24 17:01:13.356	2005-05-24 17:01:13.356	24.95	buy	s:932	132.0	133.20	17582.40

Recent Orders

### Account Profile

<a href="#">user ID:</a>	uid:255	<a href="#">full name:</a>	first:735 last:4195
<a href="#">password:</a>	●●●	<a href="#">address:</a>	354 Oak St.
<a href="#">confirm password:</a>	●●●	<a href="#">credit card:</a>	0-481-490-680
<a href="#">email address:</a>	uid:255@38.com	<input type="button" value="update_profile"/>	

Note: Click any [symbol](#) for a quote or to trade.

s:0, s:1, s:2, s:3, s:4

Trade Account	Trade3
---------------	--------

Created with IBM WebSphere Application Server and WebSphere Studio Application Developer  
 Copyright 2000, IBM Corporation

Figure 6-2 Trade scenario servlet executes random transaction

There is yet another interesting design point in the Trade application. Since it was designed as a performance test application, it uses a number of J2EE technologies and can be configured to run in JDBC or CMP EJB mode. You can find all runtime options for the application on the **Configuration** link and then **Configure Trade run-time parameters** link as shown in Figure 6-3 on page 124.

Performance Application

WebSphere Performance Benchmark Sample

WebSphere software

[Overview](#)
  
[Technical Documentation](#)
  
[Benchmarking](#)
  
[Configuration](#)
  
[Go Trade!](#)
  
[Web Primitives](#)

WebSphere software

Trade Configuration

Trade3

**Current Trade Configuration:**

The Current Trade runtime configuration is detailed below. View and optionally update run-time parameters.

**NOTE:** Parameters settings will return to default on server restart. To make configuration settings persistent across application server stop/starts, edit the servlet init parameters for each Trade servlet. This is described in the [Trade FAQ](#).

<b>Run-Time Mode</b> <ul style="list-style-type: none"> <li><input checked="" type="radio"/> EJB</li> <li><input type="radio"/> Direct</li> </ul>	Run Time Mode determines server implementation of the TradeServices to use in the Trade application Enterprise Java Beans including Session, Entity and Message beans or Direct mode which uses direct database and JMS access. See <a href="#">Trade FAQ</a> for details.
<b>Order-Processing Mode</b> <ul style="list-style-type: none"> <li><input checked="" type="radio"/> Synchronous</li> <li><input type="radio"/> Asynchronous_1-Phase</li> <li><input type="radio"/> Asynchronous_2-Phase</li> </ul>	Order Processing Mode determines the mode for completing stock purchase and sell operations. Synchronous mode completes the order immediately. Asynchronous_1-phase mode uses MDB/JMS to queue the order to a Trade broker agent to complete the order. Asynchronous_2-Phase performs a 2-phase commit over the EJB Entity/DB and MDB/JMS transactions. See <a href="#">Trade FAQ</a> for details.
<b>Access Mode</b> <ul style="list-style-type: none"> <li><input checked="" type="radio"/> Standard</li> <li><input type="radio"/> WebServices</li> </ul>	Access Mode determines the protocol used by the Trade Web application to access server side services. The Standard mode uses the default Java RMI protocol. The Web Services mode uses the WebSpheres implementation of Web Services including SOAP, WSDL and UDDI.
<b>Web Services Endpoint</b> <input type="text" value="http://localhost/trade/services/Tra"/>	For the Web Services Access mode, set the Web Services Endpoint URL to point to the host and port which is running the Trade Application Web Services module.
<b>Scenario Workload Mix</b> <ul style="list-style-type: none"> <li><input checked="" type="radio"/> Standard</li> <li><input type="radio"/> High-Volume</li> </ul>	This setting determines the runtime workload mix of Trade operations when driving the benchmark through TradeScenarioServlet. See <a href="#">Trade FAQ</a> for details.

Figure 6-3 Trade application configuration window

This will be our starting point for migration.

### 6.4.3 Migrating the sample application

In this section, we describe the steps we performed to migrate Trade. We also explain the issues and the solutions to the problems we experienced when migrating the Trade application.

#### Installing the J2EE Competitive Migrator Plugin 1.2.0 plugin

The J2EE Competitive Migrator is a plugin for Rational Application Developer version 6.0 that supports the migration of J2EE applications developed for competitive application servers to WebSphere Application Server. Currently, the tool supports the migration of EJB modules deployed for BEA WebLogic Server to WebSphere Application Server version 5.0, 5.1, or 6.0.

Refer to Appendix B, “Additional material” on page 323 for instructions on how to download the J2EE Competitive Migrator Plugin 1.2.0.

#### Importing the Trade EAR file

We start the migration of Trade by importing the EAR file into Rational Application Developer V6.

1. In Rational Application Developer V6, open the **Resource** perspective.
2. Right-click in the **Navigator** view and select **Import**.
3. In the next dialog, choose **EAR file** from the list and click **Next**.
4. Next, click **Browse** and select the Trade EAR file:  
`<source_home>\trade3w1s.ear`
5. Click **Next** and select the **trade3AppClient.jar** from the list, then click **Next**.
6. Make sure all modules are selected and click **Finish**.
7. If your workspace is not set up to build automatically, then you need to rebuild this new project now. Press **Ctrl+B** to rebuild all projects in the workspace or you can **Ctrl-Click** projects and rebuild only selected projects.

**Tip:** Rational Application Developer V6 rebuilds the project automatically after each change to source code. You can turn this off by selecting **Project** → **Build automatically** from the menu.

You should have the following projects in the Navigator view:

- ▶ trade3AppClient  
Contains a J2EE client application for Trade.
- ▶ trade3EJB  
This project contains the Trade EJBs.

- ▶ trade3Web  
The Trade Web application containing servlets and JSPs.
- ▶ trade3  
This project contains the EAR application.

## Fixing errors reported by Rational Application Developer V6

After building the whole project, open the Problems view and analyze all the errors and warnings that are reported by Rational Application Developer V6. We noticed that the Problems view reports errors in the EJB and Web application projects. We fix the errors by following these steps.

1. Open the EJB project and drill down to the folder containing the errors:  
`/trade3EJB/ejbModule/com/ibm/websphere/samples/trade/ejb`  
This folder contains both the EJB source code and compiled classes. The files that contain errors are all using BEA WebLogic Server 8.1 specific classes, so we delete all these files along with their respective class files. After deleting the files, Rational Application Developer V6 no longer reports any errors for the EJB project.
2. The Web application project also contains BEA WebLogic Server 8.1 specific classes, so we delete the following folder:  
`/trade3Web/JavaSource/jsp_servlet`
3. Rational Application Developer V6 still reports errors for the Web application and client project. This is because the projects depend on classes located in the EJB project. To fix the problem, right-click the Web application project and select **Properties**.
4. From the left pane in the properties window for trade3Web, select **Java Build Path**.
5. Select the **Project** tab and check the **trade3EJB** project in the list.
6. Click **OK**.
7. We also specify the same dependency for the client project as illustrated in Figure 6-4 on page 127.
8. If your workspace is not set to automatically rebuild, then rebuild the project manually now by pressing **Ctrl+B**.

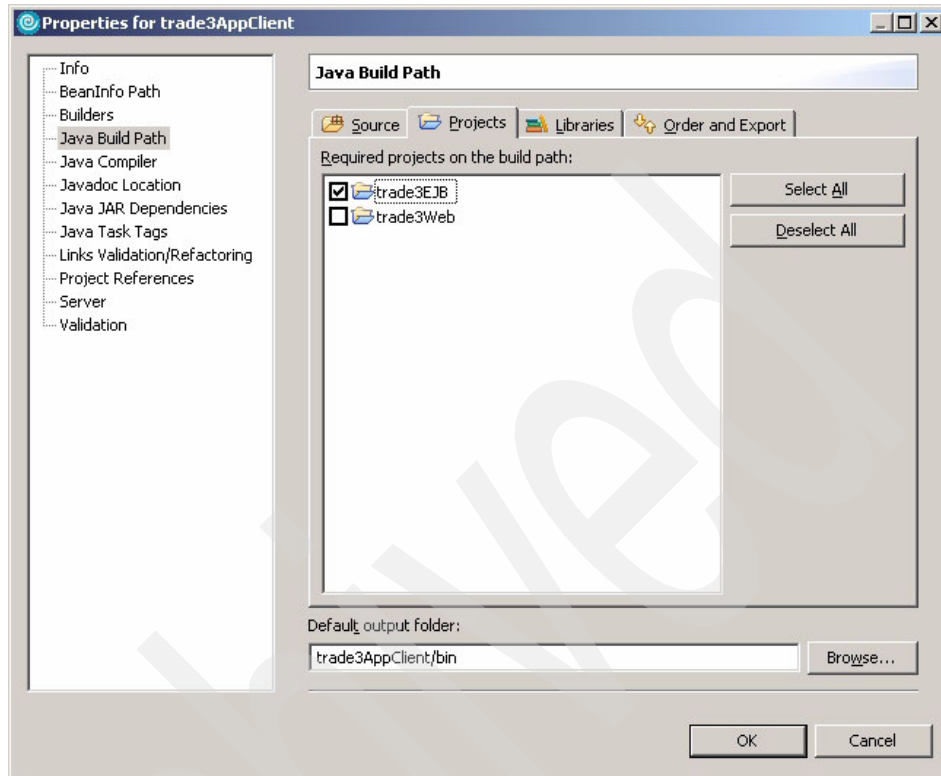


Figure 6-4 *trade3AppClient project's dependencies*

All errors reported by Rational Application Developer V6 are now fixed. But there are still some warnings, shown in Example 6-2, that we will explain how to fix next.

*Example 6-2 Warnings reported by Rational Application Developer V6*

---

IWAE0024W The Manifest Class-Path for archive trade3EJB.jar contains an entry, soap.jar, that is not resolvable to a file or module in the Enterprise Application: trade3wls.  
Broken link - /trade/OrderId

---

We fix the first warning in the list by removing the reference to soap.jar from the Jar manifest file located in Rational Application Developer V6:

/trade3EJB/ejbModule/META-INF/MANIFEST.MF

Rational Application Developer V6 now only reports the following type of errors:

- ▶ Broken links. We do not fix these errors, since they are errors in the original application.
- ▶ Serializable errors as shown in Example 6-3. These warnings will go away when we import the source code.

*Example 6-3 Errors from the Problems view in Rational Application Developer V6*

---

```
CHKJ2500I: getAllQuotes() in method java.util.Collection must be serializable
at runtime (EJB 2.0: 7.10.5)
```

---

## Importing the source code

The Trade EAR file does not include source code so we have to import it manually for each project.

**Tip:** If possible, try to include the source code in the EAR file when creating the package that will be used in the migration. This will save you the manual and error-prone task of copying the source code manually.

Import the source code for `trade3AppClient` first.

1. Copy the source code located in the directory where unzipped the `trade3-WebLogic.zip` file:  
`<source_home>/trade3AppClient`  
to:  
`trade3AppClient/src`  
located in the Rational Application Developer V6 workspace, Navigator view.
2. After importing the source code, make a clean build. Select **trade3AppClient** from the Navigator view. Then select **Project** → **Clean** from the menu and select **Clean Selected Project** and also **Start a build immediately**, then click **OK**.
3. Next, import the EJB project's source code by copying the files located in:  
`<source_home>/trade3EJB`  
to:  
`trade3EJB/ejbModule`  
in Rational Application Developer V6 workspace, Navigator view.
4. The Web module source code is imported by copying the source code from:  
`<source_home>/trade3Web`  
to:  
`/trade3Web/JavaSource`



in Rational Application Developer V6 workspace, Navigator view.

5. After importing the code, select all the Trade 3 projects, right-click the projects and click **Refresh**. Now you can build all projects by pressing **Ctrl-B**. Rational Application Developer V6 reports that it cannot find some imported classes, as shown in Example 6-4. This error is caused by a missing project that the classes depend on at compilation time. At runtime, this would not be a problem.

*Example 6-4 Missing imports*

---

```
com.ibm.websphere.samples.trade.client.ws.TradeWSServices cannot be resolved  
(or is not a valid type) for the field  
TradeWebSoapProxy.trade
```

---

6. Fix this problem by creating a new Dynamic Web Project. It will contain the missing Java source code. From the menu, select **File** → **New** → **Project**. Create a new Dynamic Web Project and name it `trade3WSAppClient`. In the advanced properties, deselect the option **Add module to an EAR project** and click **Finish**.

**Note:** Trade for WebSphere Application Server V6 contains two client applications, of which only one is included in the BEA WebLogic Server 8.1 version. The source code for the missing application is needed at runtime so we have to import the source code.

7. Copy the source code from the following directory:

```
<source_home>/trade3WSAppClient
```

to

```
/trade3WSAppClient/JavaSource
```

in Rational Application Developer V6 workspace, Navigator view and refresh the project.

8. Right-click the **trade3EJB** project and select **Properties**.
9. From the left pane in the properties window for `trade3EJB`, select **Java Build Path**.
10. Select the **Project** tab and select the **trade3WSAppClient** project in the list.
11. Click **OK**.
12. Rebuild all projects by selecting **Project** → **Build all** from the menu.

All errors reported by Rational Application Developer V6 are now fixed. There are many warnings which are not critical, for example the warning shown in Example 6-5 on page 130, but we will not fix these.

**Important:** In a real-world migration project, it is wise to try to fix all warnings for deprecated methods. This will potentially make the application easier to migrate in the future.

*Example 6-5 Deprecated methods in the Web service project*

The constructor `OperationDesc(String, ParameterDesc[], QName)` is deprecated  
`TradeWSServicesSoapBindingStub.java`

## Creating a new back end

Trade needs to be configured to use the same DB2 Universal Database V8.2 database as in the initial environment. The J2EE Competitive Migrator Plugin 1.2.0 plugin also requires that we import the database schema used by CMPs as a back end before we run it, since it will try to map the CMP fields to the database schema.

We do this with the EJB to RDB mapping wizard, which will create the necessary mapping files needed by the CMPs to use the existing database schema.

1. Open the Navigator view in the Resource perspective.
2. Right-click the **trade3EJB** project. Select **EJB to RDB mapping** → **Generate Map** from the menu.
3. Choose **Create a new backend folder** and click **Next**.
4. Choose **Meet-in-the-middle** on the next window. Click **Next**.
5. Configure the database connection using the following values:

*Table 6-9 Database connection configuration*

Parameter	Value
Connection name	Trade3 db connection
Database	trade3db
User ID	db2admin
Password	its0ral0
Database vendor type	DB2 Universal Database Express V8.2
JDBC driver	IBM DB2 APP DRIVER

6. Click **Next** and select all the **DB2ADMIN** tables on the next window.

7. Click **Next** and for “Select Meet-in-the-Middle Mapping Options” select **None**. The CMP to database mapping will be done with the The J2EE Competitive Migrator Plugin 1.2.0 plugin.
8. Click **Finish**.

The wizard creates the Map.mapxmi file, which is used to map CMP fields to table columns. The file is located along with the database schema in the following folder:

```
/trade3EJB/ejbModule/META-INF/backends/DB2EXPRESS_V82_1
```

**Important:** Verify that the back end we created is the default one. Open the ejb-jar.xml file with the Deployment Descriptor Editor. On the Overview tab, scroll all the way down and verify that the Backend ID is DB2EXPRESS\_V82\_1.

## Running the migration tool

We use the J2EE Competitive Migrator Plugin 1.2.0, since Trade already contains the BEA WebLogic Server 8.1 specific deployment descriptors.

We first tried to migrate without using the tool, instead we used the EJB to RDB mapping wizard, but ran into some problems, as shown in Example 6-6. The problem was caused by an incomplete mapping of CMPs to the database schema.

### *Example 6-6 Migration without tool*

---

```
[EJBDeploy] Error generating findByUserID(java.lang.String) query for bean
AccountEJB (Abstract schema name=Account). Error=WQRY0036E: Account a does not
have a field profile
[EJBDeploy] Query='SELECT OBJECT(a) FROM Account a WHERE a.profile.userID = ?1'
```

---

To run the migration tool, right-click the EJB project and choose **BEA Weblogic Migration** → **Migrate EJB Project**.

After running the tool, Rational Application Developer V6 will report an error for /trade3EJB/ejbModule/META-INF/weblogic-ejb-jar.xml. Fix it by deleting the file. The migration tool scans the project for BEA WebLogic Server 8.1 specific deployment descriptors and tries to migrate them to WebSphere Application Server V6.

In our case, the tool creates the ibm-ejb-jar-bnd.xmi file and maps the CMPs to the database schema. To verify this, open Map.mapxmi and enable the **Show only the unmapped objects**. The tree containing the EJBs should now only contain the top-level TradeEJBs, as illustrated in Figure 6-5 on page 132.

**Note:** J2EE Competitive Migrator Plugin 1.2.0 did not create a complete deployment descriptor for WebSphere Application Server V6. We still have to specify, for example, the JNDI names for some of the EJBs.

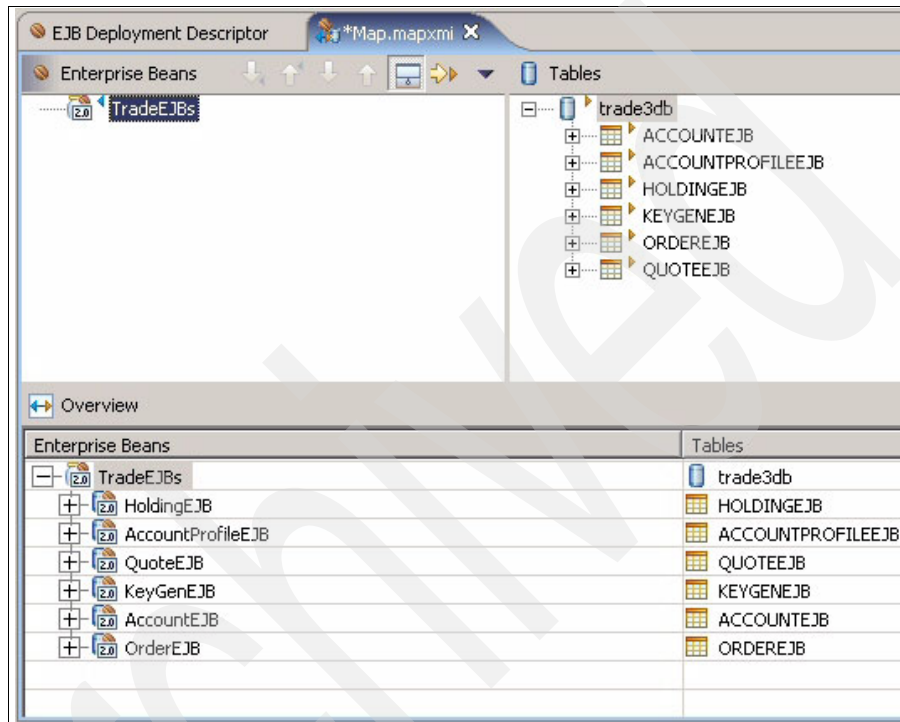


Figure 6-5 Map.mapxml file after running the migration tool. Notice the empty list to the left.

The tool performed the following tasks:

- ▶ Mapped CMPs to the database
- ▶ Mapped resource references to JNDI names
- ▶ Configured the default JDBC datasource for CMPs
- ▶ Configured the JNDI names of some of the EJBs

## Configuring WebSphere Application Server V6

In this section, we describe how to configure WebSphere Application Server V6 to run Trade. You may want to create new WebSphere profile and a new WebSphere test server based on that profile. Refer to Chapter 4, "Installation and configuration" on page 63 for further details on WebSphere Profiles creation.

To configure WebSphere Application Server V6, open a Web browser and access the Administrative Console at the following URL:

`http://localhost:9060/ibm/console`

The major difference compared to BEA WebLogic Server 8.1 is in how we configure JMS. This is because WebSphere Application Server V6 uses Service Integration Bus, which is an ESB-like approach, as the back end for the default JMS messaging engine.

**Note:** A service integration bus in WebSphere supports applications using message-based and service-oriented architectures. A bus is a group of one or more interconnected servers or server clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members. More information about SIB can be found in the WebSphere InfoCenter at the following URL:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.pmc.doc/tasks/tji0000\\_.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.pmc.doc/tasks/tji0000_.html)

## Configuring JMS

In this section, we describe how to configure the components necessary for JMS messaging. We will use the browser-based administrative GUI, but this task can also be scripted using wsadmin command line tool. This way it will take only a few moments to create all necessary resources on a new server.

### *Creating a new bus*

Before we can use JMS, we need to create the new bus:

1. Open the WebSphere Administrative Console. Select **Service Integration** → **Buses** in the navigation pane on the left.
2. On the following page, create a new ESB. Specify bus as the name. Click **OK**.
3. Save the changes you made to the configuration.

### *Creating a bus member*

The bus has to be deployed to a bus member. The members of a service integration bus are the application servers within which messaging engines for that bus can run.

1. From the WebSphere Administrative Console, select **Service Integration** → **Buses** in the navigation pane on the left. Select the bus we just created.
2. Click **Bus members** in the Additional Properties section.
3. Click **Add** and select the server to which the bus should be deployed. Click **Next** and then **Finish**.

4. Save the changes you made to the configuration.

**Tip:** Manual configuration is error-prone and sometimes difficult. Automating these tasks will save you time. You can download the WebSphere version of the Trade application together with JACL scripts for the wsadmin scripting facility. Those scripts create all necessary resources and perform application deployment in a clustered environment. Download the WebSphere version of Trade from the following URL:

<http://www.ibm.com/software/webservers/appserv/was/performance.html>

Refer to the WebSphere Information Center for more details about the administrative scripting:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.base.doc/info/aes/ae/welcadminsript.html>

### ***Creating bus destinations for JMS queue and topic***

The destination is the target for all messages sent to the Enterprise Service Bus. Follow these steps to create the bus destinations for the JMS queue and topic:

1. Select the bus we created previously.
2. Click **Destinations** in the Additional properties section. Then create a new queue destination. Specify Trade Queue as the identifier and deploy it to the bus member we created.
3. Create a Topic space in the same way as above. Specify Trade Topic Space as the identifier.
4. Save the changes.

You have now fully configured the bus.

### ***Creating a JMS connection factory***

Before configuring the JMS queues, you need to configure a JMS connection factory, which will be used by the Trade MDBs to create a connection to the JMS provider.

1. Select **Resources** → **JMS Providers** → **Default messaging** from the navigation pane on the left in the WebSphere Administrative Console.
2. Click **JMS queue connection factory** in the Connection Factories section.
3. Create a new JMS queue connection factory with the following characteristics:

Table 6-10 JMS queue connection factory configuration

Parameter	Values
Name	Trade queue connection factory
JNDI name	jms/QueueConnectionFactory
Bus name	bus

- Click **OK**, then save the changes you made.
- Next go back to **Resources** → **JMS Providers** → **Default messaging**. Click **JMS topic connection factory** in the Connection Factories section.
- Create a new JMS topic connection factory with the following characteristics:

Table 6-11 JMS topic connection factory configuration

Parameter	Values
Name	Trade topic connection factory
JNDI name	jms/TopicConnectionFactory
Bus name	bus

- Save the changes.

### Creating the JMS queue

Trade contains two Message Driven Beans. One MDB requires that you configure a new JMS queue:

- Browse to **Resources** → **JMS Providers** → **Default messaging**.
- Click **JMS queue** in the Destinations section.
- Create a new JMS queue with the following characteristics:

Table 6-12 Mail queue configuration

Parameter	Value
Name	Trade Broker Queue
JNDI name	jms/TradeBrokerQueue
Bus name	bus
Queue name	Trade Queue

- Click **OK**.

### ***Creating a JMS topic***

The second MDB in Trade requires that you configure a new JMS topic:

1. Browse to **Resources** → **JMS Providers** → **Default messaging**.
2. Click the **JMS topic** link in the Destinations section.
3. Create a new JMS topic with the following characteristics:

*Table 6-13 Mail queue configuration*

Parameter	Value
Name	Trade Topic
JNDI name	jms/TradeStreamerTopic
Bus name	bus
Topic space	Trade Topic Space

4. Click **OK**.
5. Save the changes.

### ***Creating a JMS activation specification for the JMS queues***

The last thing you need to configure in WebSphere Application Server related to JMS is the JMS activation specification which binds the MDBs to the default JMS messaging provider. The JNDI name of the JMS activation specification that you create will be used in the MDB's deployment descriptor to bind the MDB to the messaging queue.

1. Select **Resources** → **JMS Providers** → **Default messaging** from the navigation pane on the left in the WebSphere Administrative Console.
2. Click **JMS activation specification** in the Activation Specification section.
3. Create a new JMS activation specification with the following characteristics:

*Table 6-14 JMS activation specification configuration*

Parameter	Values
Name	Trade Queue Activation Specification
JNDI name	eis/tradeQueueActivationSpec
Destination type	Queue
Destination JNDI name	jms/TradeBrokerQueue
Bus name	bus



4. Create a second JMS activation specification in the same way as above with the following characteristics:

*Table 6-15 JMS activation specification configuration*

Parameter	Value
Name	Trade Topic Activation Specification
JNDI name	eis/tradeTopicActivationSpec
Destination type	Topic
Destination JNDI name	jms/TradeStreamerTopic
Bus	bus

5. Click **OK** and save your changes.

At this point, you have all the necessary components for JMS messaging configured. The next step is to configure the MDBs to use the JMS activation specification and the queues.

### ***Configuring Message Driven Beans***

Use Rational Application Developer V6 and its Deployment Descriptor Editor to edit the standard ejb-jar.xml deployment descriptor and to create the WebSphere Application Server V6 specific deployment descriptors.

1. Open the Deployment Descriptor Editor by double-clicking the ejb-jar.xml in the Navigator view. The Navigator view can be found in the Resource perspective in Rational Application Developer V6.
2. Select the **Bean** tab. You should see the same window as depicted in Figure 6-6 on page 138.

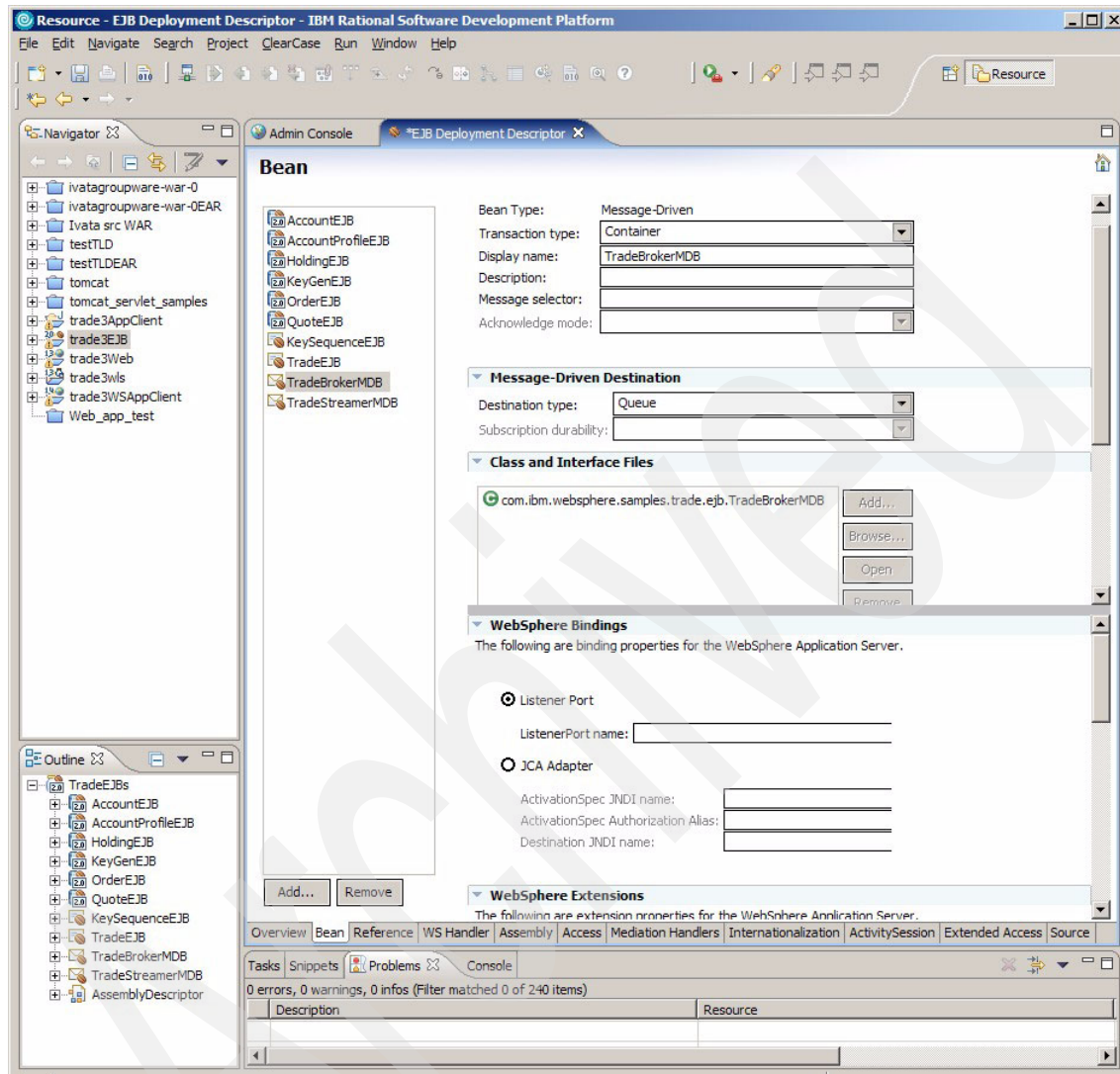


Figure 6-6 EJB Deployment Descriptor editor

3. Select **TradeBrokerMDB** from the list of EJBs. In the WebSphere Bindings section, select **JCA Adapter** and specify the following values:

Table 6-16 OrderProcessor MDB's WebSphere binding configuration

Parameter	Value
ActivationSpec JNDI name	eis/tradeQueueActivationSpec
Destination JNDI name	jms/TradeBrokerQueue

You have now configured the MDB to use the JCA Activation Specification and Destination configured in the previous section.

4. Select **TradeSteamerMDB** from the list of EJBs. In the WebSphere Bindings section, select **JCA Adapter** and specify the following values:

Table 6-17 Mailer MDB's WebSphere binding configuration

Parameter	Value
ActivationSpec JNDI name	eis/tradeTopicActivationSpec
Destination JNDI name	jms/TradeStreamerTopic

You have now completed the configuration of the MDBs and JMS messaging.

## Creating a new datasource

Trade uses a JDBC datasource for connecting to the database. To set up the datasource, you need to configure a JDBC provider, a J2C alias for storing the database authentication information and the datasource itself.

**Note:** J2EE Connector Architecture (JCA) data entries are used by resource adapters and JDBC datasources. A JCA data entry contains authentication data, which includes an Alias (an identifier), User ID (a user identity), Password and Description.

The configuration is done from the WebSphere Administrative Console.

1. Open the WebSphere Administrative Console. Click **Resources** → **JDBC Providers** and click **New** to create a new JDBC provider with the following characteristics:

Table 6-18 JDBC provider configuration

Parameter	Value
Database type	DB2
Provider type	DB2 Legacy CLI-based Type 2 JDBC Driver

Parameter	Value
Implementation type	XA datasource

- Click **Next**. Specify the correct path to the DB2 JDBC driver and click **Apply**. In our migration example, the path to the JDBC driver is:  
C:\IBM\SQLLIB\java\db2java.zip
- Next, click **Data sources** in the Additional properties section and create a new datasource with the following characteristics:

Table 6-19 JDBC datasource configuration

Parameter	Value
JNDI name	jdbc/TradeDataSource
Database name	trade3db

- Click **Apply**.
- Next, click the **J2EE Connector Architecture (J2C) authentication data entries** in the Related items section.
- Create a new alias and specify the following values:

Table 6-20 J2C alias configuration

Parameter	Value
Alias	DB2user
UserId	db2admin
Password	its0ral0

- Click **OK** and go back to the datasource you just created. From the Component-managed authentication alias drop-down menu, select the J2C alias you just created.
- Click **OK** and save your configuration.

Next, verify the connection to the database. Select the datasource you just created from the list and click **Test connection**.

## Migrating EJBs

This section describes how to configure the Session Beans and Container Managed Beans to run on WebSphere Application Server V6. In our migration example, we used the Deployment Descriptor Editor in Rational Application

Developer V6 and the migration tool mentioned previously, to generate the WebSphere Application Server V6 specific deployment descriptors.

### ***Specifying EJB JNDI names***

The first step is to assign JNDI names to the Trade EJBs, since the migration tool did not do it for all of the EJBs.

1. From the Rational workspace, open the ejb-jar.xml with the Deployment Descriptor Editor and click the **Bean** tab.
2. Select the **AccountEJB**. In the WebSphere Bindings section, enter the following value:

*Table 6-21 WebSphere bindings for the Account EJB*

Parameter	Value
JNDI name	ejb/AccountEJB

3. Repeat step two for each EJB in the list except for the MDBs. Specify a unique JNDI name for each EJB in the following format ejb/<EJBName>.

### ***Configuring the default datasource for CMPs***

Next, we specify a default datasource that will be used by all CMP beans.

1. Select the **Overview** tab in the Deployment Descriptor Editor.
2. In the WebSphere Bindings section, under the heading *JNDI - CMP Connection Factory Binding*, enter the following values:

*Table 6-22 Default datasource configuration*

Parameter	Value
JNDI name	jdbc/TradeDataSource

### **Building the application**

We used the Rational Application Developer V6 to build the migrated Trade application.

1. Right-click the **trade3wls** EAR project and select **Export**.
2. Select **EAR file** and click **Next**.
3. Choose a destination for the EAR file and press **Finish**.

### **Deploying the application**

You could deploy the application from Rational Application Developer V6 directly into the WebSphere test server with only two clicks of a mouse, but for variety,

we decided to deploy it manually using GUI. In a production environment, you would typically script this task with the wsadmin tool to achieve repeatable results in an automated fashion.

1. Make sure that WebSphere Application Server V6 is started. If you are using the WebSphere Test Environment inside of the Rational Application Developer V6, you can start WebSphere server in one of two ways:

- Open the Server view in the J2EE perspective. If you do not see a test server defined, you can right-click anywhere in the server view to create a new server. Once you have created the server, you can click the **Start** button in the Server view toolbar or right-click the server instance and select **Start**.
- You can also start the server from command line. To do so, you will need to go to command line and issue these commands:

```
cd <rad_home>\runtimes\base_v6\profiles\default\bin
startserver server1
```

2. Open the WebSphere Administrative Console:

```
http://localhost:9060/ibm/console
```

3. Log in and select **Applications** → **Install New Application** in the navigation pane to the left.
4. Specify the path to the EAR file you just created and click **Next**.
5. Select **Generate Default Bindings** on the next page. Click **Next**.
6. On the next page, there are a total of thirteen steps. We skip them all, since we have created the WebSphere Application Server V6 specific deployment descriptors. Click **Step 13** and then **Finish**.
7. Save the configuration.

**Important:** We experienced some issues while deploying the application on one of our systems. We were seeing the following messages in the deployment window (more detailed information is in the SystemOut.log file) and our deployment failed:

```
Building: /trade3EJB
Deploying jar trade3EJB
Generating deployment code
Refreshing: /trade3EJB/ejbModule.
org.eclipse.emf.common.util.BasicEList$BasicIndexOutOfBoundsExcp
tion:
index=0, size=0
at org.eclipse.emf.common.util.BasicEList.get(BasicEList.java(Compiled
Code))
at
com.ibm.etools.ejbrcdbmapping.impl.ForwardFlattenedFKComposerImpl.findColForS
impleAttribute(ForwardFlattenedFKComposerImpl.java:177)
at
com.ibm.etools.ejbrcdbmapping.impl.ForwardFlattenedFKComposerImpl.findColForA
ttribute(ForwardFlattenedFKComposerImpl.java:185)
at
com.ibm.etools.ejbrcdbmapping.impl.ForwardFlattenedFKComposerImpl.findColForA
ttribute(ForwardFlattenedFKComposerImpl.java:167)
```

This was caused by missing EJBs to RDD mappings. Repeat the steps described in “Creating a new back end” on page 130, first deleting the back end directory you created before. Make sure you save the Map.mapxml file right after you open it and check for unmapped objects. If the mappings look as shown in Figure 6-5 on page 132 then close the file and open it again and verify the mappings one more time. Then export the EAR project again.

Deploying Trade takes a couple of minutes and at the end you should see the following message: Application Trade3 installed successfully. Save the configuration and then start the Trade3 application by going to **Applications** → **Enterprise Applications** and start the Trade3 module.

The migrated Trade application is now deployed and running on WebSphere Application Server V6.

## Testing the application

The application can be accessed at the following URL:

<http://localhost:9080/trade>

We browse through the site to check that the application is running correctly. For the next set of tests, we access the following URL:

<http://localhost:9080/trade/scenario>

This URL is handled by the TradeScenarioServlet. The servlet emulates a population of users by executing an action for a randomly chosen user on each access to the URL.

The last tests we perform are found on the following page:

[http://localhost:9080/trade/web\\_prmtv.html](http://localhost:9080/trade/web_prmtv.html)

This URL contains a set of tests that we go through by clicking the links one-by-one. All tests should be executed without errors.

#### 6.4.4 Summary

We have completed the migration of the Trade application without having to change the Java code. This application was originally developed for WebSphere, then ported to WebLogic, and we have ported it back to WebSphere. Trade is a fairly complex application and uses a large number of vendor-specific deployment descriptors with exactly the same code base.

As you have seen, we have redeployed the compiled EAR file without having to change the Java code. As expected, we have not observed major migration issues, but we were able to demonstrate the following concepts:

- ▶ Use of the Competitive Migrator Plugin for Rational Application Developer V6 for automatic migration of CMP EJB mappings and vendor-specific deployment descriptors.
- ▶ Configuration of JDBC provider and datasource, Service Integration Bus, JMS queues and topics and application deployment tasks with WebSphere Application Server V6.
- ▶ Basic development activities using Rational Application Developer V6.

### 6.5 xPetstore EJB migration

xPetstore EJB is an LGPL licensed open source project hosted by SourceForge. It is a complete rewrite of Sun's PetStore J2EE reference application. xPetstore EJB was built to demonstrate J2EE development best practices and how to leverage some of the most popular open source frameworks like XDoclet and Struts. We decided to include the xPetstore EJB because it is a good example of how to write portable application that run on multiple application servers. And because it covers many interesting and widely used technologies and frameworks:

- ▶ Message Driven Beans 2.0
- ▶ Stateless Session Beans 2.0
- ▶ Stateful Session Beans 2.0



- ▶ Container Managed Persistence EJB 2.0
- ▶ JSP 1.2
- ▶ Servlet 2.3
- ▶ Ant build script
- ▶ XDoclet
- ▶ JUnitEE

xPetstore is also different from most WebSphere Application Server V6 applications in the sense that it uses XDoclet extensively to generate the following artefacts:

- ▶ EJB home and business interfaces as well as deployment descriptors.
- ▶ Application server specific deployment descriptors.
- ▶ Value objects.
- ▶ Deployment descriptors for servlets, servlet filters and JSP taglibs.

### 6.5.1 Migration approach

There are different alternatives for migrating xPetstore EJB as shown in Figure 3-1 on page 52. For xPetstore EJB, we decided to select the one covered in 3.4.4, “Alternative 4: Developing using Rational Application Developer V6” on page 56. In this migration example, we migrate the EJB version of xPetstore from BEA WebLogic Server 8.1 to WebSphere Application Server V6.

We use Rational Application Developer V6 as the development environment and a migration tool that includes a WebSphere Application Server V6 runtime environment and tools that simplify J2EE application development.

We have to create all the WebSphere Application Server V6 specific deployment descriptors since WebSphere Application Server V6 does not fully support xDoclet and vice versa. Creating all these deployment descriptors manually would be a very difficult task but by using Rational Application Developer V6 tools and wizards, this is a relatively simple task.

The migration of the xPetstore EJB application is to be done in the following sequence:

1. First, we will build the application from source code using Ant.
2. We will configure BEA WebLogic Server 8.1 to create all necessary resources for the application, such as Datasources, Mail provider, etc. We will deploy xPetstore EJB into BEA WebLogic Server 8.1 to verify that it works properly by running JUnitEE tests provided with the application.
3. Import the EAR file from WebLogic into Rational Application Developer V6 to take advantage of its advanced deployment descriptor editing functionality.

We do not change source code, but we will update resource references and export the EAR file.

4. We will configure WebSphere Application Server V6 to create the same resources we have created in BEA WebLogic Server 8.1 for xPetstore EJB application.
5. We will deploy the application into WebSphere Application Server V6 and rerun our JUnitEE test cases to make sure they are executed successfully.

**Note:** In the migration process, we remove the dependency on Ant and XDoclet. We do not modify the Ant build script to work with WebSphere Application Server V6. Rational Application Developer V6 includes all the necessary tools for making the builds. This is only done to highlight how Rational Application Developer V6 is different from using Ant and XDoclet.

It is still recommended that you have an automated build process. We recommend that the build script be created from scratch after the migration is completed successfully.

## 6.5.2 Configuring the initial environment

The initial environment consists of xPetstore running on BEA WebLogic Server 8.1 and using DB2 Universal Database V8.2 as its data store. xPetstore EJB officially only supports BEA WebLogic Server 7.x but will run on BEA WebLogic Server 8.1 without modifications. By default, xPetstore EJB does not support DB2 Universal Database V8.2, so we have included the steps required for configuring xPetstore EJB to use DB2 Universal Database V8.2.

### Getting the xPetstore EJB application

Before we start our migration, we need to obtain source files for the xPetstore EJB application. You can download file xpetstore-3.1.3.zip with all source files from the following URL:

<http://xpetstore.sourceforge.net>

In our scenario, we downloaded and extracted the application to the following directory (later referred to as <source\_home>):

D:\sampleApp\xpetstore-cmp-weblogic

### Creating the sample application database

BEA WebLogic Server 8.1 will automatically create the database tables required by the CMPs in xPetstore EJB by using the top-down approach. This is done when the application is first started. We only show how to create the database in DB2 Universal Database V8.2:

1. From a command line, start DB2 CLP (command line processor) by typing the following command:

```
db2cmd
```

2. Now, while you are in the DB2 command prompt, execute the following commands to create the xPetstore EJB database and to verify connectivity:

```
db2 create db wl_xp_ej
db2 connect to wl_xp_ej user db2admin using its0ra10
db2 disconnect all
```

At this point, the database is empty, the necessary tables will be created the first time you access the xPetstore EJB application.

### Adding the DB2 JDBC driver to the classpath

For this migration, we created a new WebLogic domain called xpetstore-cmp in order to have a clean starting environment. Therefore, we had to repeat some of the procedures already described during the migration of the Trade application. The details of how to add the DB2 drivers are covered in “Adding the DB2 JDBC driver to the WebLogic classpath” on page 118.

### Creating a new connection pool

Open the WebLogic Server Console and navigate to **xpetstore-cmp** → **Services** → **JDBC** → **Connection Pools**. Configure a new JDBC Connection Pool with the following characteristics:

Table 6-23 JDBC connection pool configuration

Parameter	Values
Database type	DB2
Database driver	IBM's DB2 Driver (Type 2 XA) Versions:7.X,8.X
Connection pool name	Petstore Connection Pool
Database name	wl_xp_ej
Username	db2admin
Password	its0ra10

### Creating a new datasource

In the WebLogic Server Console, navigate to **xpetstore-cmp** → **Services** → **JDBC** → **Data Sources**. Configure a new datasource with the following characteristics:

Table 6-24 JDBC datasource configuration

Parameter	Value
Name	Petstore JDBC Datasource
JNDI Name	jdbc/xpetstore

## Creating a new JMS server

Create a new JMS Server from the WebLogic Server Console by navigating to **Services** → **JMS** → **Servers**. Configure a new JMS server with the following characteristics:

Table 6-25 JMS Server configuration

Parameter	Value
Name	Petstore JMS Server
Target	myserver

## Creating the JMS queues

On the same page where we created the JMS server, select **Configuration** → **General**. Click **Configure Destinations** and configure a new JMS Queue with the following characteristics:

Table 6-26 Order MDB's JMS queue configuration

Parameter	Value
Name	Order queue
JNDI name	jms/queue/order

This queue will be used by the Order MDB in xPetstore EJB. The next step is to create another JMS queue to be used by the Mailer MDB. Create a new JMS queue in the same way as described earlier, specifying the following values:

Table 6-27 Mail MDB's JMS queue configuration

Parameter	Value
Name	Mail queue
JNDI name	jms/queue/mail

## Creating a JMS connection factory

Click **Services** → **JMS** → **Connection Factories** and configure a new JMS connection factory with the following characteristics:

Table 6-28 JMS connection factory configuration

Parameter	Value
Name	PetStore JMS Connection Factory
JNDI name	jms/ConnectionFactory
Target	myserver

Click **Configurations** → **Transactions** tab and select **XA Connection Factory Enabled**.

## Creating a new JavaMail session

From the WebLogic Server Console, navigate to **Services** → **Mail**. Create a new JavaMail session with the following characteristics:

Table 6-29 JavaMail session configuration

Parameter	Value
Name	PetStore Mail Session
JNDI name	mail/MailSession
Properties	mail.from=admin@localhost; mail.transport.protocol=smtp; mail.smtp.host=localhost; mail.store.protocol=pop3; mail.pop3.host=localhost; mail.user=admin
Target	myserver

## Configuring the application to use DB2

This section describes how to configure the xPetstore application to use DB2 Universal Database V8.2 since xPetstore does not support DB2 by default.

In order to use DB2 Universal Database V8.2, you have to make some changes in the configuration files of the application. This changes are necessary because the application is using Hibernate to map the application objects with the relational data, and it needs to know which database it is accessing, and some other configuration data, such as usernames and passwords.

Follow these steps to configure xPetstore Servlet to use DB2 Universal Database V8.2:

1. Edit the following file:

`<source_home>\conf\db\database.properties`

and change the *database* property to:

`database=db2`

2. In the same directory, create a file called `db2.properties` like the one shown in Example 6-7.

*Example 6-7 xPetstore database configuration file*

---

```
#=====
# DB2 configuration
#=====
db.driver=COM.ibm.db2.jdbc.app.DB2Driver
db.url=jdbc:db2:w1_xp_ej
db.user=db2admin
db.password=its0ra10
db.classpath=${lib.dir}/db2java.zip
db.foreign.key=true

hibernate.dialect=cirrus.hibernate.sql.DB2Dialect
hibernate.generator.class=native
hibernate.outer.join=true
```

---

## Building the sample application

This section describes all the necessary steps for making a build of xPetstore EJB that runs on the BEA WebLogic Server 8.1 platform. xPetstore EJB uses Ant build scripts for making builds. The build script consists of the following files:

- ▶ `<source_home>\xpetstore-ejb\build.xml`
- ▶ `<source_home>\xpetstore-ejb\build-weblogic.xml`

The build script is configured by modifying the following property files:

- ▶ `<source_home>\conf\as\appserver.properties`
- ▶ `<source_home>\conf\as\weblogic.properties`

To make the build, follow these steps:

1. Change the xPetstore EJB target platform to Weblogic by changing the `app.server` property to `weblogic` in the following file:

`<source_home>\conf\as\appserver.properties`

2. Build xPetstore EJB by running build.bat from the following directory (you may want to check what is the value of your JAVA\_HOME variable and reset it in the build.bat file if necessary):

<source\_home>\xpetstore-ejb

Ant will use the default target *all* to build the application. This makes Ant perform the following tasks:

1. Clean the build directory by removing any temporary files.
2. Run XDoclet, which generates source code and deployment descriptors.
3. Compile all source code.
4. Create a jar file containing EJBs.
5. Create a WAR file containing the Web application.
6. Create an EAR file <source\_home>\dist\xpetstore-ejb.ear containing the EJB and WAR file.

The EAR file contains the following files that make up the complete xPetstore EJB application:

- ▶ xpetstore-ejb.jar
- ▶ xpetstore-ejb.war
- ▶ xpetstore-ejb-test.war

## Deploying the sample application and populating the database

This section describes how to deploy the EAR package containing the xPetstore EJB application, to BEA WebLogic Server 8.1.

1. Copy the xpetstore-ejb.ear file to the following directory:

<weblogic\_home>\user\_projects\domains\xpetstore-cmp\applications

**Note:** BEA WebLogic Server 8.1 will automatically deploy the application if started in development mode.

BEA WebLogic Server 8.1 will also create the database schema needed by the CMPs if the WebLogic instance is running in development mode. In this case, WebLogic will add an extra column called wls\_temp to each container-created tables.

2. Populate the database with the sample xPetstore EJB data by opening a command line window and executing the following commands.

**Important:** Since our WebLogic domain was created to run in development mode, when we deploy the application it will create the tables with an extra column. Hence, the data.sql provided by the xPetstore EJB application will not work.

Refer to Appendix B, “Additional material” on page 323 for instructions on how to download the xpetstore-cmp-wls.data.zip. Extract the file into the <source\_home>xpetstore-ejb\sql directory.

The difference from the original data.sql is that this file contains the column names where the data will be inserted. This is needed since WebLogic changed the container-created tables.

```
db2cmd
db2 connect to wl_xp_ej user db2admin using its0ra10
db2 -tvf <source_home>xpetstore-ejb\sql\xpetstore-cmp-wls.data.sql
```

**Note:** The user you use to connect to the database will define the name of the schema where the ddl will create the tables.

3. Access the application at the following URL:

<http://localhost:7001/xpetstore-ejb>

At this point, you have the database with all the sample data populated; the next step is to test the application.

## Testing the application

In this section, we show you how to test the xPetstore EJB application deployed on BEA WebLogic Server 8.1. Testing is straightforward since the EJB version of xPetstore includes test cases implemented with JUnitEE.

1. To run the test cases go to the following URL:

<http://localhost:7001/xpetstore-test/index.html>

You should see the JUnitEE test page as illustrated in Figure 6-7 on page 153.

2. Select the tests options you want to test and click **Run**. On the next page, you should see no errors.



The image shows the JUnitEE TestRunner interface. It has a red header bar with the text "JUnitEE TestRunner". Below the header, there are two main sections. The first section is titled "Enter the name of a test suite to run" and contains a text input field and a "Run" button. The second section is titled "Select test suite(s) to run" and contains three checkboxes, each followed by a test suite name: ☒ xpetstore.services.cart.test.CartTest, ☒ xpetstore.services.petstore.test.PetstoreTest, and ☒ Run all tests. Below these checkboxes is another "Run" button.

Figure 6-7 JUnitEE test page for xPetstore EJB

### 6.5.3 Migrating the sample application

In this section, we describe the steps we performed to migrate xPetstore EJB. This migration effort should be straightforward since xPetstore EJB was developed following best-practices and J2EE standards.

#### Importing the application EAR file

Start the migration by importing to Rational Application Developer V6 the EAR file that was built for and deployed to BEA WebLogic Server 8.1.

**Note:** We used a new Rational Application Developer V6 workspace for the migration. The workspace folder will be referred to as <rad\_workspace>.

To import the EAR file start Rational Application Developer V6 and open the **Resource** perspective.

1. Right-click in the **Navigator** view and choose **Import** from the context menu. You should see the **Import** dialog as illustrated in Figure 6-8 on page 154.

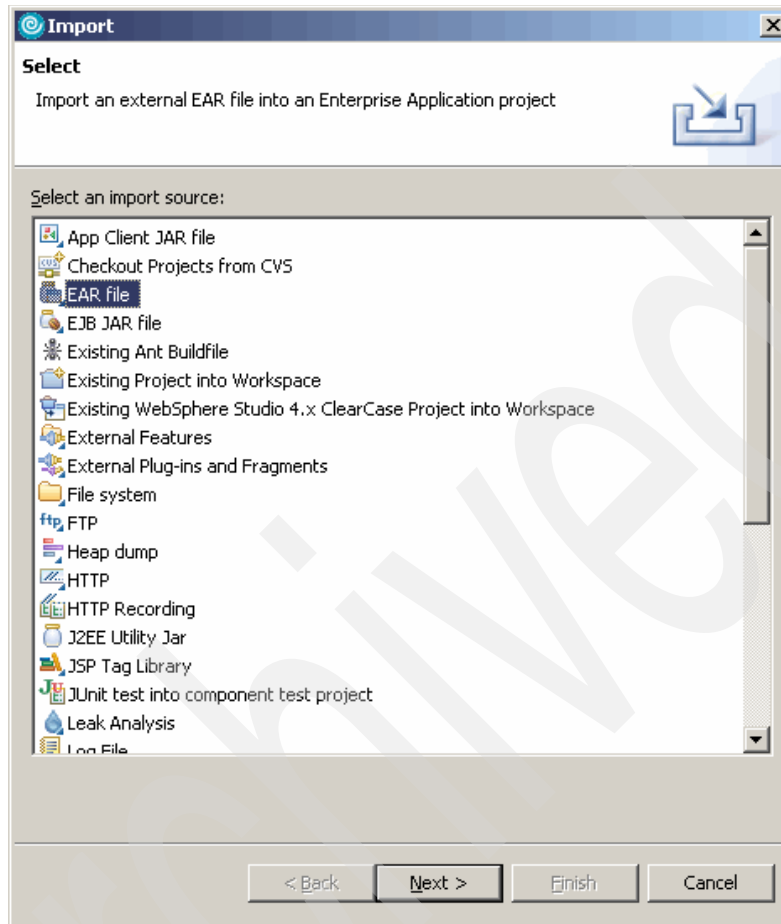


Figure 6-8 Import EAR dialog in Rational Application Developer V6

2. Select the **EAR file** option from the list and click **Next**. On the next page, specify the EAR file built for BEA WebLogic Server 8.1 located in the following directory:  
`<domain_home>\applications\xpetstore-ejb.ear`
3. Click **Finish**. It may take a few minutes to finish the import of the project and automatically perform all necessary configuration and validation actions.

The wizard automatically created four new projects in your workspace as illustrated in Figure 6-9 on page 155.

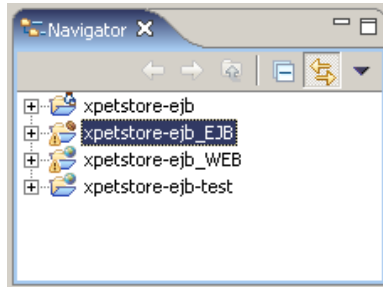


Figure 6-9 Navigator view in Rational Application Developer V6 after importing EAR file

The first item in the navigator view is the EAR project. The second item contains the EJB project and the third item the Web application project. The last item is the test project which contains unit tests.

**Tip:** Before proceeding, it is recommended that you configure Rational Application Developer V6 to not to build the projects automatically after changes to the source. This is done by deselecting the **Build Automatically** option in the Project menu.

## Creating a new back end

When you first import the EAR file into Rational Application Developer V6, the project is configured to use Cloudscape as the back end database. Our initial and destination environments use DB2 Universal Database V8.2. In the source environment, the database schema was created by BEA WebLogic Server 8.1. We will use the same database but create a new schema designed for WebSphere Application Server V6.

There are three options available in Rational Application Developer V6 for mapping CMPs to a database.

- ▶ Bottom-up  
This option generates EJBs and the mapping from an existing database.
- ▶ Top-down  
Generates a database schema and map from existing EJBs.
- ▶ Meet-in-the-middle  
Generates the mapping between an existing database and existing EJBs.

**Note:** We used the same database as BEA WebLogic Server 8.1 but will create a new schema that will be used by WebSphere Application Server.

We did not use the database schema generated by BEA WebLogic Server 8.1, since the generation of the CMP mapping files using the Meet-in-the-middle approach did not work.

This is because the table and column names do not match the CMPs. In Chapter 7, “Migrating from JBoss” on page 175, we describe how to use the meet-in-the-middle approach in more detail.

In a real world migration project, the existing database would most likely be used without modifications. For that scenario, use the Meet-in-the-middle approach.

Next, we describe how to generate a new database schema for xPetstore EJB by using the top-down approach:

1. Still in the Navigator view in the Resource perspective in Rational Application Developer V6. Right-click the **xpetstore-ejb\_EJB** project.
2. Select **EJB to RDB Mapping** → **Generate Map** from the context menu.
3. Choose **Create a new backend folder** and click **Next**.
4. Select **Top-Down** from the list of options and click **Next**.
5. Specify the following values on the next page:

*Table 6-30 Database configuration for wizard*

Parameter	Value
Target database	DB2 Universal Database Express V8.2
Database name	wl_xp_ej
Schema name	NULLID

6. Leave the remaining options by default and click **Finish**. You should be able to see the same folder structure in the Navigator view as illustrated in Figure 6-10 on page 157.

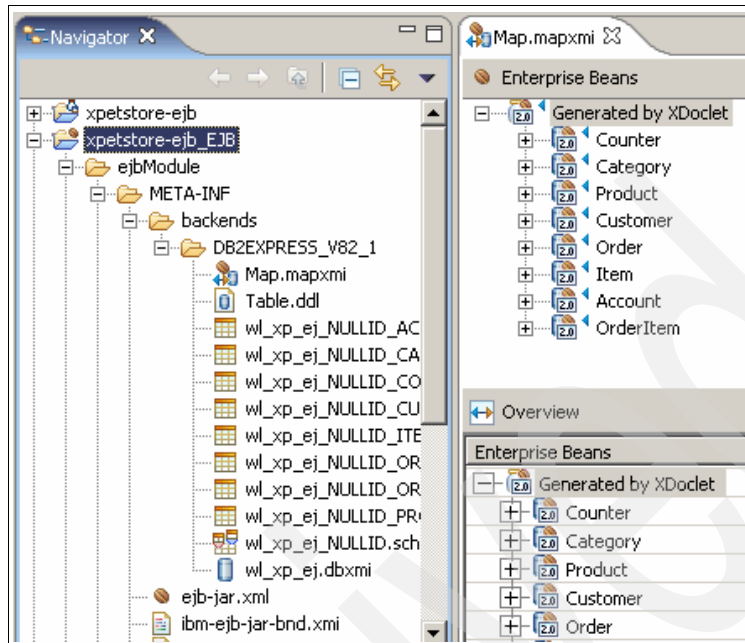


Figure 6-10 Database mapping generated with Top-down option

7. Verify that the back end is active by double-clicking the **ejb-jar.xml** file in the Navigator view. This opens the Deployment Descriptor Editor.

On the Overview tab, scroll down to the Backend ID section and verify that it is using DB2 Express V8.2 as illustrated in Figure 6-11.

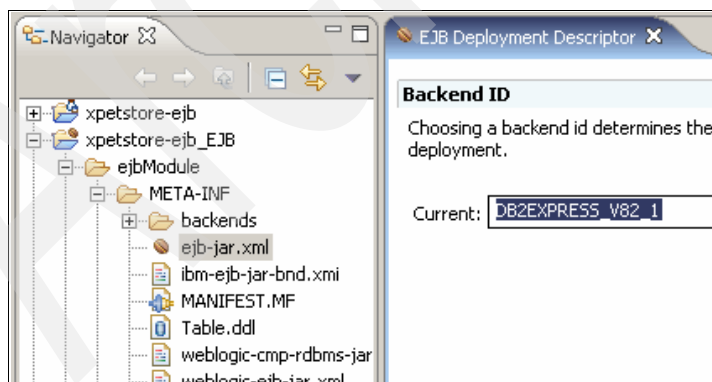


Figure 6-11 EJB Deployment Descriptor

In the following section, we will use the Table.ddl file that was generated by the wizard to create the database schema.

## Creating a new database schema

The next step is to create the database by importing the database schema that was created using the EJB to RDB Mapping wizard described previously.

Create the database schema in the wl\_xp\_ej database by following these steps:

1. Open a DB2 Command Line Processor (CLP) by executing the following command from a command line window:

```
db2cmd
```

2. From the DB2 Command Line Processor (CLP), connect to the DB2 database and create the database schema with the following commands:

```
db2 connect to wl_xp_ej user db2admin using its0ra10
db2 -tvf
<rad_workspace>\xpetstore-ejb_EJB\ejbModule\META-INF\backends\DB2EXPRESS_V8
2_1\Table.ddl
```

**Note:** DB2 Universal Database V8.2 will report the error shown in Example 6-8.

The error is caused by the CUSTOMER table which is too big to fit in the default table space. To fix this, create a new table space with a page size of at least 8 KB.

This problem can also be fixed by making the columns smaller. For example, open the file Table.ddl; do a search for “250” and replace with “150”.

### *Example 6-8 Database error*

---

```
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL0286N  A default table space could not be found with a page size of at
least "8192" that authorization ID "DB2ADMIN" is authorized to use.
SQLSTATE=42727
```

---

Now you need to import the xPetstore data. We used a modified version of the original data.sql file since the database column names have changed in this new schema.

1. Open the following file in an editor:

```
<source_home>\xpetstore-ejb\sql\data.sql
```

2. Rename all the table names to match the new schema (Table.ddl). This means removing T\_ from all table names.

3. Change the LISTPRICE and UNITCOST columns' datatype, since they are defined as DOUBLE in the new database schema and as VARCHAR in the data script.

This is done by removing the single quotes from all the **INSERT** commands for the ITEM table, so that the column values are defined as illustrated in the following example.

*Example 6-9 Updated INSERT statement*

---

```
INSERT INTO ITEM VALUES ('EST-1', 'Large', 16.50, 10.00, 'fish1.jpg',  
'FI-SW-01');
```

---

**Tip:** To perform this operation, you may want to download the GNU *sed* utility and run the following command:

```
sed "s/'\([0-9]*\.[0-9]*\)'/\1/g" data.sql > data2.sql  
rename data2.sql data.sql
```

4. Import the data by issuing the following commands from a DB2 Command Line Processor (CLP) window:

```
db2 connect to wl_xp_ej user db2admin using its0ra10  
db2 -tvf <source_home>\xpetstore-ejb\sql\data2.sql
```

## Configuring the Mail Provider in WebSphere

The JavaMail session provides an abstraction layer for communicating with an e-mail server. This section describes how to configure a Mail Provider which is used to create a JavaMail session in WebSphere.

1. Start the WebSphere Application Server V6 by typing the following commands:

```
cd <was_home>\profiles\<profile_name>\bin  
startserver server1
```

2. Open the WebSphere Administrative Console by accessing the following URL:

```
http://localhost:9060/ibm/console
```

3. Select **Resources** → **Mail Providers** from the left navigation pane.
4. Click the **Mail Providers** → **Built-in Mail Provider** link. Click **Mail Sessions** in the Additional Properties section. Create a new mail session with the following characteristics:

Table 6-31 JavaMail session configuration

Parameter	Value
Name	xPetstoreMail
JNDI name	mail/xpetstore/MailSession
Mail transport host	localhost
Mail transport protocol	SMTP
Mail store host	localhost
Mail store protocol	POP3

- Click **OK** to create the mail session.

We describe more in detail how to configure the Mailer MDB to use the JavaMail session in “Configuring Message Driven Beans” on page 165.

**Tip:** You can also create the JavaMail session using the Built-in Mail Provider which can be found in the WebSphere Administrative Console by browsing through **Resources** → **Mail Providers** → **Built-in Mail Provider**.

## Creating a new datasource

xPetstore uses a JDBC datasource for connecting to the database. To set up the datasource, we need to configure a JDBC provider, a J2C alias for storing the database authentication information and the datasource itself.

- Open the WebSphere Administrative Console and browse through **Resources** → **JDBC Providers** then click **New**. Create a new JDBC provider with the following characteristics:

Table 6-32 JDBC provider configuration

Parameter	Value
Database type	DB2 Legacy CLI-based Type 2 JDBC Driver
Implementation type	XA datasource

- Click **Next** and specify the correct path to the DB2 JDBC driver and click **Apply**. In our migration example, the path to the JDBC driver is:

C:/IBM/SQLLIB/java/db2java.zip



**Note:** We need to use an XA capable datasource because xPetstore EJB uses JMS and accesses the database at the same time. If we do not use an XA capable datasource, we will get an error as shown in Example 6-10 when we submit an order on the checkout page.

*Example 6-10 Transaction error if not using XA datasource*

```
0000003b RegisteredRes E   WTRN0063E: An illegal attempt to commit a one phase
capable resource with existing two phase capable resources has occurred.
0000003b RegisteredRes E   WTRN0086I: XAException encountered during prepare
phase for transaction
00000103FA7E431B000000001000000105FED8BF51F42A168B1FE8AF33B707D115C0EF37C0000010
3FA7E431B000000001000000105FED8BF51F42A168B1FE8AF33B707D115C0EF37C00000001.
Local resources follow.
0000003b RegisteredRes E   WTRN0089I: XATransactionWrapper@ 6909bce4
XAResource:
com.ibm.ws.sib.api.jmsra.impl.JmsJcaRecoverableSiXaResource@543bfce4 enlisted:
true mcWrapper.hashCode()582909156: Vote: commit.
0000003b RegisteredRes E   WTRN0089I: LocalTransactionWrapper@:5801bce4
LocalTransaction:com.ibm.ws.rsadapter.spi.WSRdbSpiLocalTransactionImpl@5577bce4
enlisted:true registeredForSynctruemcWrapper.hashCode()1321958628: Vote: none.
```

- Click **Data sources** in the Additional properties section and create a new datasource with the following characteristics:

*Table 6-33 JDBC datasource configuration*

Parameter	Value
Name	Petstore Datasource
JNDI Name	jdbc/xpetstore
Database name	wl_xp_ej

- Click **Apply** and save your changes.
- Click **J2EE Connector Architecture (J2C) authentication data entries** in the Related items section and create a new authentication alias with the following characteristics:

*Table 6-34 J2C alias configuration*

Parameter	Value
UserId	db2admin
Password	itsOral0

6. Click **OK** and go back to the datasource you just created. From the Component-managed authentication alias drop-down menu, select the J2C alias you just created.
7. Click **OK** and save your configuration.
8. Verify the connection to the database by selecting the datasource you just created from the list and click **Test connection**.

## Configuring JMS

WebSphere Application Server V6 comes with a new messaging engine that is based on the Enterprise Service Bus (ESB) concept. This new engine is used as the default JMS provider in WebSphere Application Server V6.

In this section, we describe how to configure the ESB components necessary for JMS messaging, after which we explain how to configure the JMS connection factory and JMS queues.

### *Creating a new service integration bus*

The first step in configuring JMS messaging in WebSphere Application Server V6 is to create an service integration bus.

1. Open the WebSphere Administrative Console and select **Service Integration** → **Buses** in the navigation pane on the left.
2. Create a new bus. Specify **bus** as the bus name and click **OK**.
3. Save the changes to the configuration.

### *Creating a bus member*

The bus has to be deployed to a bus member which is an instance of WebSphere Application Server V6. Deploying the bus to a bus member will create a messaging engine in that WebSphere Application Server V6 instance. The bus member has its own data store for messages.

1. From the WebSphere Administrative Console select **Service Integration** → **Buses** in the navigation pane on the left. Select the bus you just created.
2. Click **Bus members** in the **Additional Properties** section.
3. Click **Add** and select the server where the bus will be deployed to. Click **Next** and then **Finish**.
4. Save the changes you made to the configuration.

### *Creating a bus destination*

The destination is the target of all messages sent to the Information Service Bus. To create the bus destination, follow these steps:

1. Select the bus you just created.

2. Click **Destinations** in the Additional Properties section. Then create a new Queue destination. Specify xPetstore Queue as the identifier and deploy it to the bus member you created.
3. Click **Finish** and then save your configuration.

You have now configured the messaging bus, bus member and bus destinations, which together form the infrastructure for JMS messaging. The next step is to configure the JMS connection factory and queues.

### ***Creating a JMS connection factory***

Before configuring the JMS queues, you need to configure a JMS connection factory, which will be used by the xPetstore EJB MDBs to create a connection to the JMS provider.

1. Select **Resources** → **JMS Providers** → **Default messaging** from the navigation pane on the left in the WebSphere Administrative Console.
2. Click **JMS queue connection factory** in the Connection Factories section.
3. Create a new JMS queue connection factory with the following characteristics:

*Table 6-35 JMS queue connection factory configuration*

Parameter	Values
Name	xPetstore JMS connection factory
JNDI name	jms/xpetstore/QueueConnectionFactory
Bus name	bus

4. Click **OK**, then save the changes you made.

### ***Creating JMS queues***

xPetstore uses one Message Driven Bean for order processing and one for sending e-mails. The MDBs require that we set up two JMS queues.

1. Go to the **Resources** → **JMS Providers** → **Default messaging** page.
2. Click the **JMS queue** in the Destinations section.
3. Create a new JMS queue with the following characteristics:

*Table 6-36 Mail queue configuration*

Parameter	Value
Name	xPetstore Mail Queue

Parameter	Value
JNDI name	jms/queue/mail
Bus name	bus
Queue name	xPetstore queue

- Click **OK** and create the second JMS queue in the same way as we described previously with the following characteristics:

*Table 6-37 Order queue configuration*

Parameter	Value
Name	xPetstore Order Queue
JNDI name	jms/queue/order
Bus name	bus
Queue name	xPetstore Queue

### ***Creating a JMS activation specification for the JMS queues***

The last thing you need to configure in WebSphere Application Server, related to JMS, is the JMS activation specification that will bind the MDBs to the default JMS messaging provider. The activation specification is new for J2EE 1.4.

The JNDI name for the JMS activation specification that you create is used in the MDB's deployment descriptor to bind the MDB to the messaging queue.

- From the WebSphere Administrative Console, select **Resources** → **JMS Providers** → **Default messaging** from the navigation pane on the left.
- Click **JMS activation specification** in the Activation Specifications section.
- Create a new JMS activation specification with the following characteristics:

*Table 6-38 Order MDB's JMS activation specification configuration*

Parameter	Values
Name	xPetstore Order Activation Specification
JNDI name	eis/xPetstoreOrderActivation
Destination JNDI name	jms/queue/order
Bus name	bus

- Click **OK** and save your changes.

5. Create a second JMS activation specification in the same way as described in the previous step with the following characteristics:

*Table 6-39 Mailer MDB's JMS activation specification configuration*

Parameter	Value
Name	xPetstore Mail Activation
JNDI name	eis/xPetstoreMailActivation
Destination JNDI name	jms/queue/mail
Bus	bus

You now have all the necessary components for JMS messaging configured. The next step is to configure the MDBs to use the JMS activation specification and queues.

### **Configuring Message Driven Beans**

In the previous section, we described how to configure all the server-side components related to JMS. In this section, we describe how to use the Deployment Descriptor Editor in Rational Application Developer V6, to configure the MDBs to use those components. This is done by specifying the correct JMS Activation Specification and JMS queues for each MDB in the project.

The Deployment Descriptor Editor will be used to edit the standard ejb-jar.xml deployment descriptor and to create the WebSphere Application Server V6 specific deployment descriptors. The ejb-jar.xml and the BEA WebLogic Server 8.1 specific deployment descriptors were originally generated by XDoclet.

**Note:** WebSphere Rapid Deployment (WRD) can be used to generate deployment descriptors from selected XDoclet tags.

1. Open Rational Application Developer V6 and browse to the EJB project xpetstore-ejb\_EJB.
2. Open the Deployment Descriptor Editor by double-clicking the **ejb-jar.xml** in the Navigator view from the Resource perspective.
3. Select the **Bean** tab. You should see a similar window to the one depicted in Figure 6-12 on page 166.

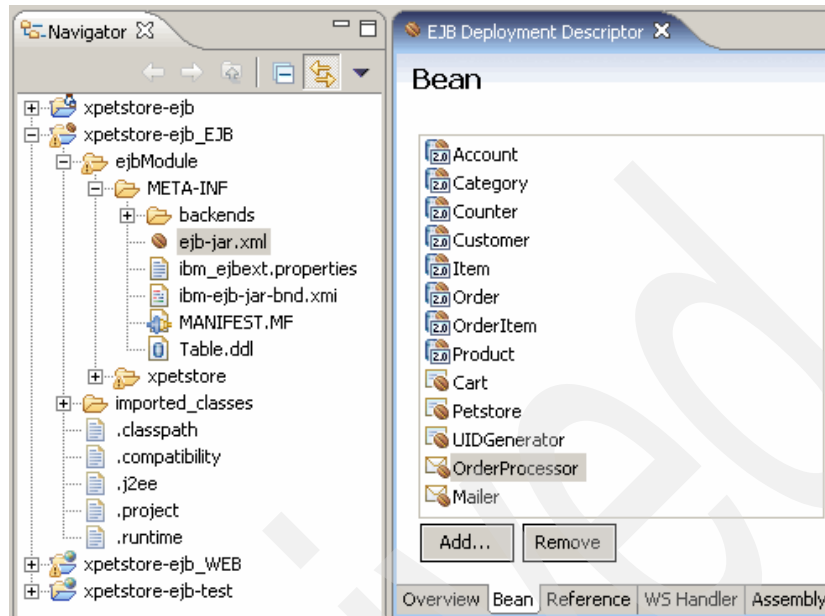


Figure 6-12 EJB Deployment Descriptor Editor

4. Select the **OrderProcessor** MDB from the list of EJBs. In the WebSphere Bindings section, select **JCA Adapter** and specify the following values:

Table 6-40 OrderProcessor MDB's WebSphere binding configuration

Parameter	Value
ActivationSpec JNDI name	eis/xPetstoreOrderActivation
Destination JNDI name	jms/queue/order

You have now configured the MDB to use the JCA activation specification and Destination you configured in the previous section.

**Tip:** Every time you make changes in the Deployment Descriptor, save those updates by pressing **Ctrl+S**.

5. Select the **Mailer** MDB in the list of EJBs. In the WebSphere Bindings section, select **JCA Adapter** and specify the following values:

Table 6-41 Mailer MDB's WebSphere binding configuration

Parameter	Value
ActivationSpec JNDI name	eis/xPetstoreMailActivation
Destination JNDI name	jms/queue/mail

You have now completed the configuration of the MDBs and JMS messaging.

## Configuring Session Beans and Container Managed Beans

This section describes how to configure the Session Beans and Container Managed Beans to run on WebSphere Application Server V6. The only things you have to change are the application specific deployment descriptors that were generated by XDoclet for BEA WebLogic Server 8.1. In our migration example, we used the Deployment Descriptor Editor in Rational Application Developer V6 to generate the WebSphere Application Server V6 specific deployment descriptors.

### Specifying EJB JNDI names

The first step is to assign JNDI names to the EJBs. This needs to be done because the EJB lookups will be done through resource references specified in web.xml and not the JNDI name directly.

1. Open the Deployment Descriptor Editor by double-clicking the **ejb-jar.xml** in the Navigator view.
2. Click the **Bean** tab. You should see a similar window to the one shown in Figure 6-12 on page 166.
3. Select the **Account** EJB and specify the following value in the WebSphere Bindings section.

Table 6-42 WebSphere bindings for the Account EJB

Parameter	Value
JNDI name	ejb/AccountLocal

4. Repeat the previous step for each EJB in the list except for the MDBs. Specify a unique JNDI name for each EJB in the following format:

```
ejb/<BeanName>Local
```

### Configuring the default datasource for CMPs

Next, we specify a default datasource that will be used by all CMP beans.

1. Select the **Overview** tab in the Deployment Descriptor Editor.

2. In the WebSphere Bindings section, under the JNDI - CMP Connection Factory Binding section, specify the following value:

Table 6-43 Default datasource configuration

Parameter	Value
JNDI name	jdbc/xpetstore

### Configuring EJB resource references

xPetstore EJB requires a JDBC datasource, JMS queues and factories that are all managed by WebSphere Application Server V6. The links to these resources are declared in the ejb-jar.xml and WebSphere Application Server specific deployment descriptors using resource references.

1. Open the Reference tab in the Deployment Descriptor Editor.

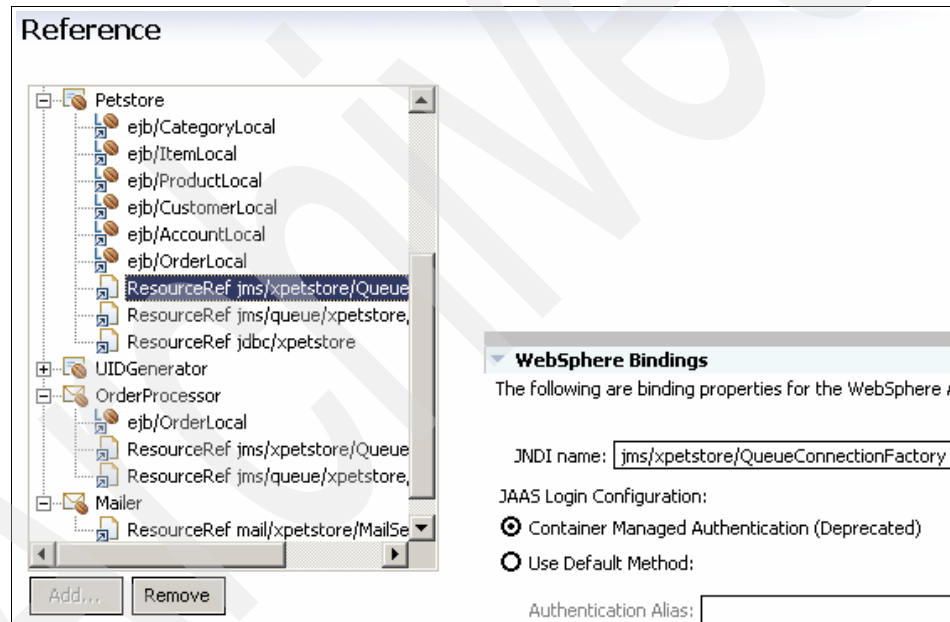


Figure 6-13 Resource reference page in the EJB Deployment Descriptor editor

2. Expand the **Petstore** EJB by clicking the plus sign next to it. Select the **ResourceRef jms/xpetstore/QueueConnectionFactory** from the list and specify the following value in the WebSphere Bindings section:



Table 6-44 Petstore EJB's WebSphere bindings

Parameter	Value
JNDI name	jms/xpetstore/QueueConnectionFactory

3. Select the **ResourceRef jms/queue/xpetstore/order** from the list and specify the following value in the **WebSphere Bindings** section:

Table 6-45 Order EJB's WebSphere bindings

Parameter	Value
JNDI name	jms/queue/order

4. Expand the **OrderProcessor** bean, select the **ResourceRef jms/xpetstore/QueueConnectionFactory** and specify the following value in the WebSphere Bindings section:

Table 6-46 OrderProcessor EJB's WebSphere bindings

Parameter	Value
JNDI name	jms/xpetstore/QueueConnectionFactory

5. Expand the **OrderProcessor** bean and select the **ResourceRef jms/queue/xpetstore/mail** then specify the following value in the WebSphere Bindings section:

Table 6-47 OrderProcessor EJB's WebSphere bindings

Parameter	Value
JNDI name	jms/queue/mail

6. Expand the **Mailer** bean and select the **ResourceRef mail/xpetstore/MailSession** then specify the following value in the WebSphere Bindings section:

Table 6-48 Mailer EJB's WebSphere bindings

Parameter	Value
JNDI name	mail/xpetstore/MailSession

## Importing the source code

xPetstore EJB was built following best practices and uses separate packages for the EJB and Web application projects, which makes it easier to import the source code.

xPetstore uses XDoclet extensively to generate source code and deployment descriptors. The deployment descriptors were included in the EAR file.

Now we describe how to import the Java source files that were not generated by XDoclet.

1. Copy the EJB project's source code folder located in the following directory:  
`<source_home>\xpetstore-ejb\java\xpetstore`  
to the xpetstore-ejb\_EJB project's `ejbModule\xpetstore` folder in the Rational Application Developer V6 workspace. You can use the command line, copy and paste, drag and drop or any other tool to copy files.
2. Copy the WAR project's source code folder located in the following directory:  
`<source_home>\xpetstore-ejb\web\xpetstore`  
to the xpetstore-ejb\_WEB project's `JavaSource\xpetstore` folder in the Rational Application Developer V6 workspace.
3. Copy the test project's source code folder located in the following directory:  
`<source_home>\xpetstore-ejb\test\xpetstore`  
to the xpetstore-ejb-test project's `JavaSource\xpetstore` folder in the Rational Application Developer V6 workspace.
4. In the Navigator view, select all four projects, right-click and select **Refresh**. This will refresh the project structure from the file system and display files you just copied.
5. Click **Project** → **Build All** in the menu. Rational Application Developer V6 now reports errors in the **Problems** view for the WAR and test projects because it cannot find the source code for the EJBs.
6. Fix the errors by specifying that the project depends on the EJB project. Right-click the **xpetstore-ejb\_WEB** project and choose **Properties** from the menu.
7. From the Properties window, select **Java Build Path** → **Projects** and select the xpetstore-ejb\_EJB project from the list. Finish by clicking **OK**.
8. The test project also references code located in the EJB project so do the same for the test project.
9. Click **Project** → **Build All** from the menu. The Problems view in Rational Application Developer V6 now contains only warnings about imports that are not used.

10. Fix these import warnings by opening the files and pressing **Ctrl+Shift+O** to automatically reorganize the imports. Save all files and rebuild.
11. You should still see two broken links warnings; we will not fix these since they are trivial.

XDoclet automatically generates the source code for EJB local and remote interfaces and the value objects when we create an xPetstore EJB build. This source code must be imported separately since it is located in a different folder.

1. Open the ejbModule folder in the EJB project. For each class file in the folder, verify that there is a corresponding Java source file in the same directory. If the source file is missing, it means that the files generated by Ant using XDoclet needs to be copied from the following directory:

```
<source_dir>\xpetstore-ejb\build\java
```

to the xpetstore-ejb\_EJB EJB project's corresponding directories.

2. Once finished, click **Project** → **Build All** in the menu. You should see no errors in the Problems view, only warnings about imports that are not used.
3. Fix these import warnings by opening the files and pressing **Ctrl+Shift+O** to automatically reorganize the imports. Save all files and rebuild.

The xPetstore EJB source code is now imported into the new development environment. You can now remove all XDoclet tags from the source code and create a new Ant script to automate the build script.

**Note:** The project structure that was created when you imported the EAR file and source code into Rational Application Developer V6 is not optimal. The EJB project's source code and compiled classes reside in the same folder. We recommend you reorganize the project structure.

## Building the application

We used the Export wizard in Rational Application Developer V6 to create the build for the migrated xPetstore EJB.

1. Right-click the **xpetstore-ejb** EAR project and select **Export** from the menu.
2. From the Export context menu, select the EAR file and click **Next**.
3. Specify a destination for the EAR file and select all the check boxes, then click **Finish**.

## Deploying the application

You could use the built-in WebSphere Application Server V6 test server in Rational Application Developer V6 to deploy the migrated application. For this

particular example, we decided to use the external server for which we have configured a new profile and defined all resources previously.

1. Start WebSphere server by using the command:

```
<was_home>\profiles\xPetStoreProfile\bin\startserver server1
```

2. Open the WebSphere Administrative Console at the following URL:

```
http://localhost:9060/ibm/console
```

3. Log in and select **Applications** → **Install New Application** in the navigation pane to the left.
4. Specify the path to the EAR file you created and click **Next**.
5. Select **Generate Default Bindings** and click **Next**.
6. On the next page, there are a total of 13 steps. Skip them all since we have created the WebSphere Application Server V6 specific deployment descriptors with Rational Application Developer V6. Click **Step 13: Summary** and then click **Finish**.
7. Once the deployment is complete without errors, save the configuration.
8. Now go to **Applications** → **Enterprise Applications** and start the xPetstore - EJB application.

**Note:** You can override the default bindings specified in the WebSphere Application Server V6 specific deployment descriptors in Step 6, 7 and 8. You can also deploy the application without the WebSphere-specific deployment descriptors and specify the resource references at the time of deployment by going through all thirteen steps in the deployment wizard.

## Testing the application

Test the application as described in “Testing the application” on page 152. All test cases should pass and you should obtain the same test results as in the initial environment:

```
http://localhost:9080/xpetstore-test/index.html
```

You can also log in into the application and see how it works:

```
http://localhost:9080/xpetstore-ejb
```

## 6.5.4 Summary

While migrating the xPetstore EJB application, we were able to redeploy it from BEA WebLogic Server 8.1 into the WebSphere Application Server V6 without changing a single line of Java code. We only had to update the deployment

descriptors, including the mappings of CMP EJBs to the database. This is a sound statement for J2EE portability.

xPetstore EJB is a J2EE 1.3 application and uses a wide spectrum of J2EE APIs, including EJB 2.0, JSP 1.2, Servlet 2.3. WebSphere Application Server V6 is a J2EE 1.4 compliant server, but we were able to run the J2EE 1.3 application in the J2EE 1.4 compliant server without changing a single line of code. This is good evidence of J2EE upward compatibility.

Some of the lessons and best practices learned in this migration are discussed in more detail in Appendix A, “Development tips for portable applications” on page 295.

Since we now have a working application and our build process was done by Rational Application Developer V6, we no longer use all those XDoclet tags that are still left in the source code. Therefore, it may be better to remove all those tags to avoid future confusion. On the other hand, there is a WebSphere Rapid Deployment (WRD) capability in WebSphere Application Server V6 that uses the same approach for artefact generation as XDoclet and, in fact, the tags are almost identical. If we had wanted to use WRD as our preferred development approach then we might have selected a different migration approach as is documented in 3.4.2, “Alternative 2: Migrating XDoclet tags to WRD” on page 53.



## Migrating from JBoss

In this chapter, we describe how to migrate two J2EE 1.3 applications from JBoss 3.2.7 to WebSphere Application Server V6. At the time of writing this book, the latest generally available version of JBoss was 4.0.2. version 4.0 of JBoss shipped in late 2004, but since we were searching for sample applications on the Internet (mostly from the <http://sourceforge.net> site), we realized that a majority of applications are written for JBoss 3.x and lower.

There are several challenges associated with migrating applications from open source runtimes. One of them is the fact that these applications are often built on top of different open source frameworks and libraries, such as Apache Commons, Struts and many others. Naturally there is a concern that those frameworks may not work on commercial applications servers. We demonstrate in this book that several frameworks that were used in our sample applications work very well with WebSphere Application Server. For more detailed description of frameworks and how they were used in our sample applications, please refer to Table 1-2 on page 4.

## 7.1 Introduction

JBoss 3.2.7 is an open source implementation of Sun's J2EE 1.3 specification, but it was never J2EE 1.3 certified. One example of JBoss 3.2.7 not implementing all of the J2EE 1.3 requirements is the lack of support for Client Container. Otherwise, JBoss 3.2.7 has a fairly advanced implementation of EJB 2.0 and JMX. There are parts of the J2EE 1.3 specification that JBoss does implement only partially, for example the lack of a persistent transaction log for JTA. Additionally, the implementation of JMS is immature in JBoss 3.2.7, although it is being rewritten for the future JBoss 5 release. JBoss 3.2.7 also bundles Apache Tomcat as part of its distribution. Therefore, all content discussed in Chapter 8, "Migrating from Tomcat" on page 257 is also relevant here. For this reason, we will mostly focus on EJB migration and will not pay much attention to JSP and servlet migration issues.

The xPetstore EJB 3.1.3 application that we have selected for migration does not use any non-J2EE features of JBoss, but there is a possibility that your application may have some dependency on JBoss AOP or other components that are not portable to other application servers and may require a redesign or rewrite.

This is a good time to mention Sun's Java Application Verification Kit (AVK) for the Enterprise as a tool intended to help developers test their applications for correct use of J2EE APIs and portability across J2EE compatible application servers, and avoid inadvertently writing non-portable code. See more details about AVK at the following URL:

[http://java.sun.com/j2ee/verified/avk\\_enterprise.html](http://java.sun.com/j2ee/verified/avk_enterprise.html)

Sun's AVK tool is not perfect, but it may come in handy when you try to quickly analyze your application for J2EE compliance.

The xPetstore EJB 3.1.3 for JBoss has the same functions as Sun's Petstore and it also has a version for WebLogic, which was used in Chapter 6, "Migrating from BEA WebLogic" on page 111. This application also has a non-EJB implementation called xPetstore Servlet which is based on Hibernate for persistence. We used this non-EJB version to illustrate Tomcat migration in Chapter 8, "Migrating from Tomcat" on page 257.

xPetstore EJB application is built using XDoclet tags. We will migrate this application from JBoss 3.2.7 into WebSphere Application Server V6 using two different methods. In the first example, we will use the approach described in 3.4.2, "Alternative 2: Migrating XDoclet tags to WRD" on page 53. In the second example, we will migrate this very same application, but this time using the method described in 3.4.3, "Alternative 3: Using XDoclet support for WebSphere" on page 55.



## 7.2 Prerequisites and assumptions

There are some prerequisites and assumptions we need to consider before carrying on with the migration examples. One basic requirement is that the reader should have the “destination” environment installed and configured as described in Chapter 4, “Installation and configuration” on page 63. This should provide some initial background knowledge if the reader is not yet familiar with IBM products.

The reader should also be familiar with the following technologies before starting migration:

- ▶ Sun J2EE 1.3 or 1.4 specification

Obviously, we assume familiarity with the Java language. We also recommend that the reader become familiar with JSP and servlet specifications as well as the Web application archive file structure (WAR files).

The following software should be installed before starting the migration:

- ▶ J2SE 1.4

We use J2SE 1.4 since it is supported by WebSphere Application Server V6 and we need to make sure our applications work well in this JVM. The WebSphere Application Server V6 installation includes JDK 1.4.2 so you do not need to install additional JDKs on your machine.

- ▶ Ant 1.5.1

Apache Ant is a Java-based build tool. It is similar to the Make tool, but has many enhanced features that make it a very good choice for Java applications. This is required for migrating the xPetstore Servlet application.

- ▶ IBM WebSphere Application Server V6

Before you start migrating to WebSphere Application Server V6, we recommend that you play with sample applications shipped with the product and also read publications listed in “Related publications” on page 325. Instructions on how to install and configure WebSphere Application Server V6 are covered in Chapter 4, “Installation and configuration” on page 63.

- ▶ IBM UDB DB2 8.2

WebSphere Application Server V6 supports many databases, but we have decided to use DB2 for all our applications. Instructions on how to install and configure DB2 Universal Database V8.2 can be found in Chapter 4, “Installation and configuration” on page 63.

- Rational Application Developer V6

Rational Application Developer V6 was only used for the migration project covered in 7.4, “xPetstore EJB migration using Alternative 2” on page 179. Instructions on how to install and configure Rational Application Developer V6 can be found in Chapter 4, “Installation and configuration” on page 63.

## 7.3 Software installation

This section provides very high-level steps and tasks for installing and configuring Tomcat, Maven, Ant and some other products to be able to run our sample applications in their original environment. Instructions on how to install and configure the WebSphere destination environment are covered in detail in Chapter 4, “Installation and configuration” on page 63.

### 7.3.1 JBoss

Detailed instructions for installing, configuring and managing JBoss 3.2.7 are provided in the product documentation guides available at the following URL:

<http://jboss.com/products/jbossas/docs>

The following list highlights the general tasks we performed to install and configure our initial environment as a starting point for deploying sample applications.

1. Download the JBoss 3.2.7 binary distribution file named jboss-3.2.7.zip ( the size is 57 MB) from the following URL:

<http://www.jboss.com/products/jbossas/downloads>

2. Unzip the archive into the directory of your choice (later referred to as <jboss\_home>).

3. In the <jboss\_home>\bin directory, create the file setenv.bat to set the JAVA\_HOME as shown below:

```
set JAVA_HOME=<was_home>\java
```

4. Create a new copy of the JBoss server. You will use two separate configurations for migrating the xPetstore application using two different alternatives.

```
Copy recursively <jboss_home>\server\default to  
<jboss_home>\server\xpetstore_xx.
```

5. You are now ready to start the JBoss 3.2.7 server. Go to your <jboss\_home>\bin directory and run the following command:

```
setenv.bat
```

```
run.bat -c xpetstore_xx
```

6. Once the server is started, you can verify that it is running by opening a Web browser and pointing it to this URL: <http://localhost:8080>. You should see the JBoss Welcome window and can click the JMX console URL.

### 7.3.2 Apache Ant

Apache Ant is a Java-based build tool widely used in Java and J2EE projects. Most of the applications we worked with, while writing this book, were using Apache Ant to create their ear or war files. Instructions on how to download, install and configure Ant 1.5.1 can be found at the following URL:

<http://ant.apache.org>

Make sure you define the Ant binaries directory in your path to be able to run Ant from your source directory. We will refer to the **ant** command as if it were defined in the path.

## 7.4 xPetstore EJB migration using Alternative 2

xPetstore EJB is a reimplementaion of the Sun PetStore blueprint application, but although it is similar in function, it is very different in implementation. The approach taken by developers of xPetstore EJB was annotation-based programming and thus the application is built using XDoclet for artefact generation. This application is distributed with the full source code under the terms of LGPL (Lesser GNU General Public license).

We found this to be an interesting application to be migrated for the following reasons:

- ▶ It is based on popular open source frameworks.
- ▶ It is designed for running on JBoss and WebLogic.
- ▶ It has not been tested on WebSphere Application Server.
- ▶ The build process uses Ant, a very popular build tool for Java.
- ▶ The application is built using XDoclet and its source code is not tied to any specific IDE. XDoclet is used to generate EJB 2.0 files, Home and Business interfaces (local and remote), ejb-jar.xml, application server specific deployment descriptors and Value Objects classes. XDoclet is also used to generate Web deployment descriptors for servlets, Web filters, JSP taglibs and Struts.
- ▶ The application uses J2EE 1.3 features like CMP 2.0 and CMR.
- ▶ Test cases for xPetstore EJB are written using JUnitEE framework.

Overall, we believe this application demonstrates good programming practices and covers a broad range of J2EE and other Java technologies.

### 7.4.1 Migration approach

The xPetstore EJB application was built with non-IBM development tools using Ant as a build tool and various editors for creating Java and XML files, mostly generated by XDoclet and other frameworks. In a typical enterprise or ISV scenario, such an application will have to potentially be enhanced over time and it will likely need to support JBoss as well as WebSphere as deployment targets. The decision has to be made as to what tool to use to migrate the application and what tools will be used to continue the development of such an application going forward. Different alternatives for migrating this application are shown in Figure 3-1 on page 52.

For this particular migration project, we have decided to use the method described in section 3.4.2, “Alternative 2: Migrating XDoclet tags to WRD” on page 53. This is certainly not the only possible approach and it has its advantages and drawbacks, but we wanted to provide an illustration and instructions on how to use this particular migration alternative. This does not mean that this is the best possible approach to migrate this application.

In this migration example, we will not migrate the entire application due to current limitations of the WRD implementation in WebSphere, but rather we will illustrate the migration approach and demonstrate the migration of a servlet, two CMP Beans and one Session Bean. The second part of this chapter will cover a full application migration. Please refer to 7.5, “xPetstore EJB migration using Alternative 3” on page 228 for details.

As to the development environment, instead of using plain text editors or Eclipse, we decided to use Rational Application Developer V6, since it has code assist for WRD as well as a built-in WebSphere Test Environment and is likely to provide more productivity compared to a plain text editor approach.

**Note:** WRD can also be used outside of Rational Application Developer V6 in a so-called “free-form” project structure. More details on this topic are discussed at the following URL:

<http://publib.boulder.ibm.com/infocenter/ws60help/topic/com.ibm.etools.wrd.freeform.doc/topics/tfreeform.html?resultof=%22%66%72%65%65%66%6f%72%6d%22%20>

The migration is performed in the following steps:

1. Verify that application works in its original environment. In this case, we will deploy and run the application in JBoss 3.2.7.

2. Import EJB java files into Rational Application Developer V6.
3. Analyze and fix problems reported, replace or edit XDoclet tags with WRD syntax as needed.
4. Build xPetstore EJB for WebSphere Application Server V6 using Rational Application Developer V6 (all deployment descriptors will be generated by WRD).
5. Deploy the application on WebSphere Application Server V6; identify and solve any problem reported.

## 7.4.2 Configuring the initial environment

It is important to make sure that the application works properly in the source environment before we try to migrate it to WebSphere. This section describes how we configured xPetstore EJB to run on JBoss 3.2.7. Instructions on how we installed the JBoss 3.2.7 environment are described in 7.3, “Software installation” on page 178.

### Installation and configuration prerequisites

Unfortunately, the original documentation for xPetstore EJB 3.1.3 is only for JBoss 3.0 and does not include the correct steps for configuring the application with JBoss 3.2.7. We have used original instructions available at the following URL as general guidance, but had to make changes as described next.

[http://xpetstore.sourceforge.net/setup\\_jboss-3.0.x.html](http://xpetstore.sourceforge.net/setup_jboss-3.0.x.html)

The following instructions represent the steps we followed in order to get up and running with xPetstore EJB for JBoss 3.2.7.

1. We have already copied a default JBoss configuration into <jboss\_home>\server\xpetstore\_a2 as described in 7.3.1, “JBoss” on page 178.
2. Now you need to configure JMS resources for the application in JBoss. You need to define two queues for orders and mail. Edit the following file and add the queue definitions as illustrated in Example 7-1.

<jboss\_home>\server\xpetstore-a2\deploy\jms\jbossmq-destinations-service.xml

*Example 7-1 Add queue definitions to jbossmq-destinations-service.xml file*

---

```
<server>
. . .
<mbean code="org.jboss.mq.server.jmx.Queue"
  name="jboss.mq.destination:service=Queue,name=order">
  <attribute name="JNDIName">queue/order</attribute>
  <depends optional-attribute-name="DestinationManager">
    jboss.mq:service=DestinationManager
```

```

    </depends>
  </mbean>

  <mbean code="org.jboss.mq.server.jmx.Queue"
    name="jboss.mq.destination:service=Queue,name=mail">
    <attribute name="JNDIName">queue/mail</attribute>
    <depends optional-attribute-name="DestinationManager">
      jboss.mq:service=DestinationManager
    </depends>
  </mbean>
</server>

```

---

- Initially, the xPetstore EJB was configured for the HSQL database, but we will use DB2 Universal Database V8.2. To configure the JDBC resources, add the DB2 JDBC driver to the classpath. Copy the db2java.zip file from:

```
<db2_home>\IBM\SQLLIB\java
```

to the following directory:

```
<jboss_home>\server\xpetstore_a2\lib
```

- To create a new datasource for the xPetstore EJB application, you need to create a new db2-ds.xml file in the following directory and edit it as described in Example 7-2.

```
<jboss_home>\server\xpetstore_a2\deploy\db2-ds.xml
```

---

*Example 7-2 Define datasource for DB2 Universal Database V8.2 in JBoss*

---

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- ===== -->
<!-- JBoss Server Configuration ===== -->
<!-- Datasource config for DB2 using xxx driver -->
<!-- ===== -->
<datasources>
  <local-tx-datasource>
    <jndi-name>jdbc/xpetstoreDS</jndi-name>
    <connection-url>jdbc:db2:petstore</connection-url>
    <driver-class>COM.ibm.db2.jdbc.app.DB2Driver</driver-class>
    <user-name>db2admin</user-name>
    <password>its0ra10</password>
  </local-tx-datasource>
</datasources>

```

---

Remember that the name of the database in DB2 Universal Database V8.2 may not be longer than eight characters. When you deploy xPetstore EJB into JBoss 3.2.7, the container will create all tables automatically (this is configurable), so you do not need to generate the schema and create those tables manually.

5. Original instructions for xPetstore EJB suggest configuring a mail service in JBoss, but we found that the default configuration works properly and there is no need to change anything.

At this point, you have finished configuring resources in JBoss and you are ready to build the xPetstore EJB application.

## Getting the xPetstore EJB application

Before starting with the migration, you need to obtain source files for the xPetstore EJB application. You can download file xpetstore-3.1.3.zip with all source files from the following URL:

<http://xpetstore.sourceforge.net/download.html>

In our scenario, we downloaded and extracted the application to the following directory (later referred to as <source\_home>):

D:\sampleApp\xpetstore-jboss

## Updating the source code for the xPetstore EJB application

Before you build the application, you need to make a few changes to the source code.

1. Open the following file:

```
<source_home>\xpetstore-ejb\java\xpetstore\domain\order\ejb\OrderEJB.java
```

Search for the string `target=multiple="yes"` and update it, replacing the first `"="` sign with the `"-"` sign:

```
target-multiple="yes"
```

2. We had problems with the deployment of the application into JBoss and had to correct some of the XDoclet tags for the application to work. In the same file, search for the string:

```
schema="Order"
```

and replace it with:

```
schema="OrderBean"
```

Also replace the string:

```
query="SELECT OBJECT(o) FROM Order AS o WHERE o.customer.userId = ?1"
```

with:

```
query="SELECT OBJECT(o) FROM OrderBean AS o WHERE o.customer.userId = ?1"
```

It is unknown at this point why JBoss refused to accept the original EJBQL, but this change fixed the problem we encountered during deployment.

## Building the xPetstore EJB application

Since we are changing the database from HSQL to DB2 Universal Database V8.2, it is also necessary to update the build environment.

1. Create the application database and tables. Assuming that the DB2 is up and running, from a command line window, type the following:

```
db2cmd
```

Now, while you are in the DB2 command prompt, execute the following commands to create the xPetstore EJB database and to verify connectivity:

```
db2 create db petstore
db2 connect to petstore user db2admin using its0ra10
db2 disconnect all
```

2. Create a file to set the environment variables, as follows:

```
<source_home>\xpetstore-ejb\setenv.bat
```

Edit the file and add the following two lines:

```
set JAVA_HOME=<was_home>\java
set JBOSS_HOME=<jboss_home>
```

Note that we are using the IBM JDK provided with WebSphere Application Server V6 to build this application and also to run JBoss 3.2.7, to make sure that the application works on this JDK in the source environment and thus to minimize change.

3. Open the file <source\_home>\conf\as\appserver.properties and make sure you have the app.server property set as follows:

```
app.server=jboss
```

4. You also need to update the build environment to reflect your current JBoss install path. Open the following file:

```
<source_home>\conf\as\jboss.properties
```

Edit the configuration parameters with the following values:

```
. . .
jboss.deploy.dir=${jboss.home.dir}/server/xpetstore_a2/deploy
. . .
jboss.datasource=java:/jdbc/xpetstoreDS
. . .
jboss.remove.table=false
. . .
jboss.typemapping.db2=DB2
. . .
```

5. Configure the proper type of mapping for JBoss. Open the following file:

```
<source_home>\xpetstore-ejb\build-jboss.xml
```

Add these lines as shown in **bold**:



```

<!-- JBoss type mappings -->
<condition property="jboss.typemapping" . . . >
. . .
</condition>

<condition property="jboss.typemapping" value="${jboss.typemapping.db2}">
  <equals arg1="${database}" arg2="db2" />
</condition>

```

6. Since the application will be using DB2, it is necessary to update the Ant variable in the <source\_home>\conf\db\database.properties file as follows:  
database=db2
7. In the same directory, create a new file db2.properties file with the following content.

*Example 7-3 DB2 configuration file*

---

```

#=====
# DB2 SQL configuration
#=====
db.driver=COM.ibm.db2.jdbc.app.DB2Driver
db.url=jdbc:db2:petstore
db.user=db2admin
db.password=its0ra10
db.classpath=${lib.dir}/main/com.ibm/db2java.zip
db.foreign.key=true

hibernate.dialect=cirrus.hibernate.sql.DB2Dialect
hibernate.generator.class=sequence
hibernate.outer.join=true
hibernate.scrollable.recordset=false

```

---

8. Create a new directory <source\_home>\lib\main\com.ibm and copy file db2java.zip into that directory.
9. Set the JAVA\_HOME variable and run the Ant script to build the application. From a command line, type the following commands:

```

cd <source_home>\xpetstore-ejb
setenv.bat
build.bat

```

Assuming that the build is completed without errors, the resulting ear file will be located in the following directory:

```
<source_home>\dist\xpetstore-ejb.ear
```

You are now ready to deploy the application into JBoss. You may want to look at some other options for the build script and their usage at the following URL:

<http://xpetstore.sourceforge.net/run.html>

## Deploying the application

Once you have installed all prerequisites and built the application EAR file, you can deploy the application. JBoss 3.2.7 will automatically deploy applications located in the <jboss\_home>\server\xpetstore\deploy\ directory by default. Follow these steps to deploy the xPetstore EJB application:

1. Start JBoss 3.2.7 server for the xpetstore\_a2 configuration that we have created:

```
cd <jboss_home>\bin
setenv.bat
run -c xpetstore_a2
```

**Tip:** We have created file setenv.bat earlier during JBoss installation. You only need to run it once while in the command line window, or you may decide to set JAVA\_HOME as part of your Windows environment variables.

2. To deploy the application, simply copy the EAR file from:

```
<source_home>\dist\xpetstore-ejb.ear
```

to the following directory:

```
<jboss_home>\server\xpetstore\deploy
```

In the console where you started JBoss, you should be able to see messages about the successful deployment of your application.

### *Example 7-4 JBoss 3.2.7 console output from deployment of the xPetstore EJB application*

---

```
. . .
21:03:44,077 INFO [EARDeployer] Init J2EE application:
file:/C:/Software/jboss-3.2.7/server/xpetstore/deploy/xpetstore-ejb.ear
21:03:45,629 INFO [EjbModule] Deploying Counter
21:03:45,649 INFO [EjbModule] Deploying Category
21:03:45,659 INFO [EjbModule] Deploying Product
21:03:45,679 INFO [EjbModule] Deploying Customer
21:03:45,689 INFO [EjbModule] Deploying Order
21:03:45,699 INFO [EjbModule] Deploying Item
21:03:45,699 INFO [EjbModule] Deploying Account
21:03:45,719 INFO [EjbModule] Deploying OrderItem
21:03:45,719 INFO [EjbModule] Deploying Cart
21:03:45,729 INFO [EjbModule] Deploying Petstore
21:03:45,729 INFO [EjbModule] Deploying UIDGenerator
21:03:45,739 INFO [EjbModule] Deploying OrderProcessor
21:03:45,739 INFO [EjbModule] Deploying Mailer
```

```

21:03:51,688 INFO [EJBDeployer] Deployed:
file:/C:/Software/jboss-3.2.7/server/xpetstore/tmp/deploy/tmp15906xpetstore-ejb.ear-contents/xp
etstore-ejb.jar
21:03:51,708 INFO [TomcatDeployer] deploy, ctxPath=/xpetstore-test,
warUrl=file:/C:/Software/jboss-3.2.7/server/xpetstore/tmp/deploy/tmp15906xpetstore-ejb.ear-cont
ents/xpetstore-ejb-test.war/
21:03:51,828 INFO [TomcatDeployer] deploy, ctxPath=/xpetstore-ejb,
warUrl=file:/C:/Software/jboss-3.2.7/server/xpetstore/tmp/deploy/tmp15906xpetstore-ejb.ear-cont
ents/xpetstore-ejb.war/
21:03:52,299 INFO [SignOnFilter] init()
21:03:52,299 INFO [SignOnFilter] ...signon.action=/signon.jspa
21:03:52,299 INFO [SignOnFilter] ...Adding URI to protect: checkout.jspa
21:03:52,299 INFO [SignOnFilter] ...Adding URI to protect: order.jspa
21:03:52,379 INFO [PropertyMessageResources] Initializing, config='org.apache.s
truts.util.LocalStrings', returnNull=true
21:03:52,379 INFO [PropertyMessageResources] Initializing, config='org.apache.s
truts.action.ActionResources', returnNull=true
21:03:52,479 INFO [PropertyMessageResources] Initializing, config='Resources',
returnNull=true
21:03:52,499 INFO [EARDeployer] Started J2EE application:
file:/C:/Software/jboss-3.2.7/server/xpetstore/deploy/xpetstore-ejb.ear

```

---

3. Once the application is deployed and started, JBoss will automatically create all database tables for CMP EJBs. You can verify that tables were created if you start DB2 Control Center by clicking **Start** → **All Programs** → **IBM DB2** → **General Administration Tools** → **Control Center**.

Once the Control Center has started, click **All Databases** → **PETSTORE** → **Tables**. You should see all tables for all the schemas; at this point, you can add a filter to display only those tables for the DB2ADMIN schema as illustrated in Figure 7-1 on page 188.

**Tip:** To use a filter, click **View** → **Filter** to show only tables that belong to the DB2ADMIN schema and hide all other tables.

Alternatively, you can run a query from a command line by typing the following commands:

```

db2cmd
db2 connect to petstore user db2admin using its0ra10
db2 list tables for schema db2admin

```

You should see a similar window with the following information.

*Example 7-5 db2 list tables command output*

```

E:\>db2 list tables for schema db2admin
Table/View      Schema  Type Creation time
-----

```

T_ACCOUNT	DB2ADMIN	T	2005-06-28-15.29.15.914000
T_CATEGORY	DB2ADMIN	T	2005-06-28-15.29.19.429001
T_COUNTER	DB2ADMIN	T	2005-06-28-15.29.17.256001
T_CUSTOMER	DB2ADMIN	T	2005-06-28-15.29.20.501001
T_ITEM	DB2ADMIN	T	2005-06-28-15.29.18.307001
T_ORDER	DB2ADMIN	T	2005-06-28-15.29.17.777001
T_ORDER_ITEM	DB2ADMIN	T	2005-06-28-15.29.20.140001
T_PRODUCT	DB2ADMIN	T	2005-06-28-15.29.16.905001

8 record(s) selected.

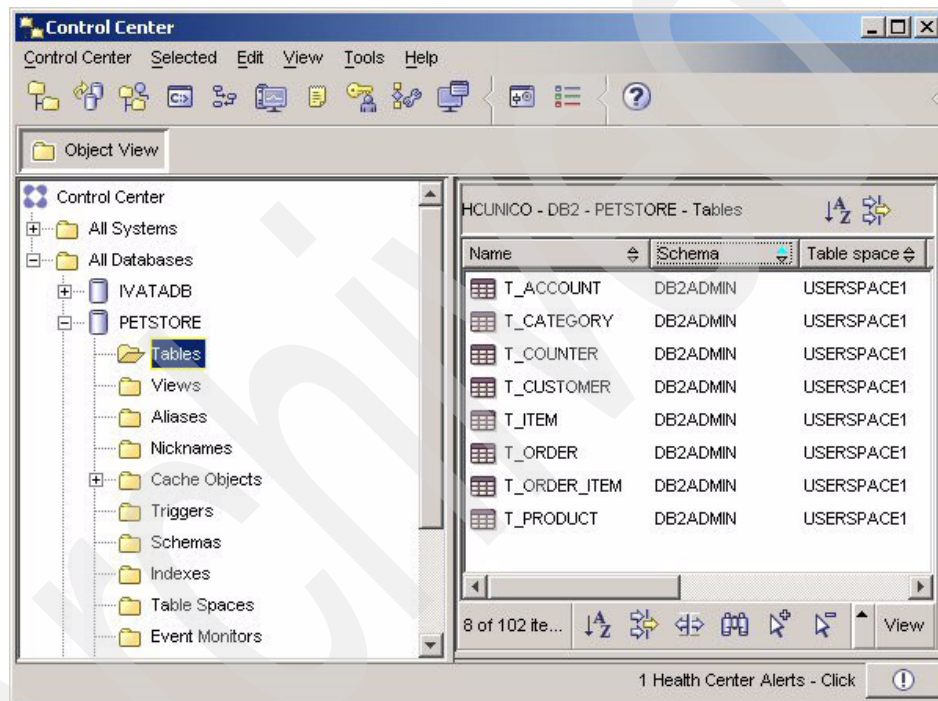


Figure 7-1 xPetstore EJB tables automatically created by JBoss deployment

- If you need the database schema in the future, you may generate a Data Definition Language (DDL) for this database using the built-in DDL export in the DB2 Control Center. To generate the DDL, you need to select all the tables, right-click any of the selected tables and select **Generate DDL** as shown in Figure 7-2 on page 189.

ROMAN41 - DB2 - PETSTORE - Tables			
Name	Schema	Table space	Comments
T_ACCOUNT	DB2ADMIN	USERSPACE1	
T_CATEGORY	DB2ADMIN	USERSPACE1	
T_COUNTER	DB2ADMIN	USERSPACE1	
T_CUSTOMER	DB2ADMIN	USERSPACE1	
T_ITEM	DB2ADMIN	USERSPACE1	
T_ORDER	DB2ADMIN	USERSPACE1	
T_ORDER_ITEM	DB2ADMIN	USERSPACE1	
T_PRODUCT	DB2ADMIN	USERSPACE1	

Figure 7-2 Generating DDL for the Petstore database

From the Generate DDL window, on the Statement tab, deselect all options, except for the **Database objects** and click **Generate**. In the next window, click **Save** and save the file to the following directory:

```
<source_home>\petstore-ejb\sql\petstore_db2.sql
```

Click **Close** in the Show Command window.

- Having created the tables, the next step is to populate the database with the initial set of data. In order to do so, you have to make some changes to the data.sql file provided with the xPetstore EJB distribution located in the following directory:

```
<source_home>xpetstore-ejb\sql\data.sql
```

The problem is that the SQL insert statements have quotes around double values that do not work with DB2.

```
INSERT INTO T_ITEM VALUES ('EST-1', 'Large', '16.50', '10.00',
'fish1.jpg', 'FI-SW-01');
```

Thus, you need to remove quotes around those double values, as shown:

```
INSERT INTO T_ITEM VALUES ('EST-1', 'Large', 16.50, 10.00, 'fish1.jpg',
'FI-SW-01');
```

You can fix these entries manually, but it may be easier to use the **sed** command (which can be downloaded for Windows). Therefore, we run the following command:

```
cd <source_home>\xpetstore-ejb\sql
sed "s/'\([0-9]*\.[0-9]*\)'/\1/g" data.sql > data_db2.sql
```

- You can now run the script to populate the database with the initial data. From a command line window, type the following commands:

```
cd <source_home>\xpetstore-ejb\sql
db2cmd
```

This command will open a new command line window with a DB2 Command Window environment set. Once you are in this new DB2 Command Window, you need to run the following DB2 commands to populate the database:

```
db2 connect to petstore user db2admin using its0ra10
db2 -tvf data_db2.sql
```

You should see all SQL statements execute successfully, as shown in the following example.

*Example 7-6 Excerpt after populating the petstore database with initial sample data*

---

```
Database Connection Information
Database server      = DB2/NT 8.1.5
SQL authorization ID = DB2ADMIN
Local database alias = PETSTORE

DELETE FROM T_COUNTER
SQL0100W No row was found for FETCH, UPDATE or DELETE; or the result of a
query is an empty table.  SQLSTATE=02000
. . .
INSERT INTO T_ACCOUNT VALUES ( 'user1', 'password1' )
DB20000I The SQL command completed successfully.
. . .
INSERT INTO T_CUSTOMER VALUES ( 'user1', 'firstname1', 'lastname1',
'herve@localdomain', '111-1111', 'en', 'street1.1', 'street1.2', 'city1',
'ST1', 'A1B-1C1', 'US', '111-111-111', 'Visa', '01-11', 'user1' )
DB20000I The SQL command completed successfully.
. . .
INSERT INTO T_PRODUCT VALUES ('K9-P0-02', 'Poodle', 'Cute dog from France',
'DOGS')
DB20000I The SQL command completed successfully.
. . .
INSERT INTO T_ITEM VALUES ('EST-11', 'Spotted Male Puppy', 48.50, 32.00,
'dog2.jpg', 'K9-P0-02')
DB20000I The SQL command completed successfully.
. . .
INSERT INTO T_COUNTER VALUES('Customer', 1000)
DB20000I The SQL command completed successfully.
```

---

**Tip:** In the xPetstore EJB configuration, JBoss properties are configured in such a way that the tables are automatically created by JBoss when the application is deployed and dropped when application is undeployed.

You may change this behavior by editing the `jboss.properties` file located in the following directory:

```
<source_home>\conf\as\jboss.properties
```

with the following values:

```
# Always create tables at each deployment
jboss.create.table=false
jboss.remove.table=false
```

Remember to rebuild and redeploy the application EAR file with new settings.

## Testing the application

Now that you have the xPetstore EJB application running and have populated the database, you can test the application by accessing it at the following URL:

```
http://localhost:8080/xpetstore-ejb
```

Once you verify that the application works properly, shut down the JBoss instance. To shut down the server, simply use the **Ctrl-C** sequence in the console in which JBoss was started. Alternatively, you can use the **shutdown.bat** command found in the `<jboss_home>\bin` directory.

## Summary

At this point, you have successfully built, deployed and tested the xPetstore EJB application on JBoss 3.2.7 runtime. Having the application working properly on the source environment is our starting point for migrating it into WebSphere Application Server V6. We also tried to minimize environment changes between JBoss 3.2.7 and WebSphere Application Server V6 by using the IBM JDK provided by WebSphere Application Server for the application build process and for running it in JBoss. We also verified that the application works properly with DB2 Universal Database V8.2. This should simplify the migration to some degree since we narrowed the scope of change.

### 7.4.3 Migrating the application

This section describes all the steps necessary for migrating the xPetstore EJB application from the JBoss 3.2.7 environment to WebSphere Application Server V6. We were not able to do a complete migration using this alternative due to some limitations in the current version of WRD shipped with WebSphere

Application Server V6. Therefore, we migrate portions of the application such as servlets, CMP EJBs and Session EJBs.

WRD also supports annotation-based programming for Message Driven Beans and Web services. WRD functionality is growing to include compatibility for all XDoclet tags.

**Tip:** The J2EE 1.5 draft specification supports metadata facility in the Java programming language (JSR 175). The strategic direction for J2EE 1.5 is to allow programmers to develop software more easily and faster without having to deal with many artefacts for each component. XDoclet played a big role in this change. Draft EJB 3.0 specification is more developer friendly than its predecessors and is one step closer to pure POJO programming model.

## Configuring DB2 Universal Database V8.2

We have already migrated the xPetstore EJB configuration from using HSQL to DB2 Universal Database V8.2 on the original environment. No additional configuration is necessary. We will try to use tables and data that were created during the deployment into JBoss.

## Migrating CMP EJB using Rational Application Developer V6

In this migration example, we will be using Rational Application Developer V6 for editing and compiling the source code and building war, jar and ear files. We will also use the built-in WebSphere Application Server Test Server that is installed along with Rational Application Developer V6.

To begin your migration, you need to create the EJB project and then import source code from xPetstore EJB application.

1. Start Rational Application Developer V6 and define a new workspace so you will have a clean environment to start migrating the application. This step is optional.
2. Turn off automatic build feature by clicking **Project** → **Build automatically**.
3. Create a new xPetstore-EJB project by clicking **File** → **New** → **EJB project**. Complete the wizard as shown in the following figure and click **Finish**. Make sure you check the property **Add support for annotated Java classes**. This is what tells the tool to add the WRD builder and recognize all tags in the Java source files.



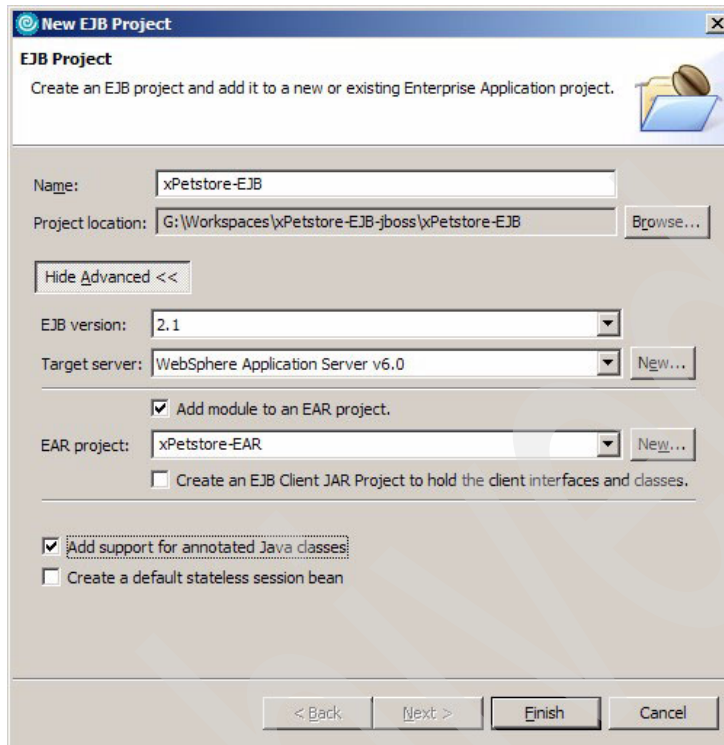


Figure 7-3 Using new EJB project wizard to create xPetstore-EJB project

**Note:** Make sure you uncheck the **Create an EJB Client JAR Project** option. Failing to do this will result in the creation of a client project and the migration will not succeed.

Once created, if you right-click the new EJB project and select **Properties**, you should see Annotation builder listed under Builders as shown in Figure 7-4 on page 194. This is what parses the WRD tags and generates code and deployment descriptors for annotations.

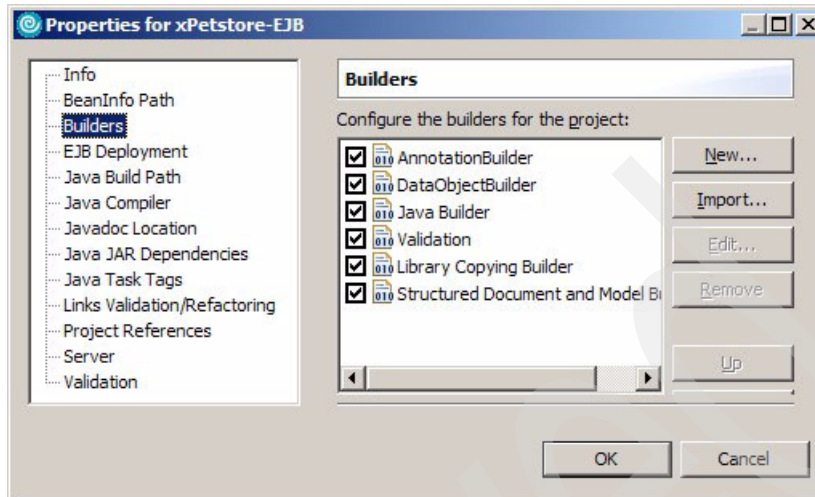


Figure 7-4 EJB project properties

4. Before you import the source code into this project, you need to define the properties of the database mapping generation for the CMPs. Right-click the EJB project you just created and select **New** → **Other** → **Simple** → **File** and click **Next**. Select the parent folder for your EJB project (in our case **xPetstore-EJB**), for the File Name type database.properties and click **Finish**.

The file automatically opens for editing; copy and paste the content of this example into that file and save the changes by pressing **Ctrl+S**.

*Example 7-7 Content of the database.properties file*

```
# This properties file provides values for aspects of the mapping generation
# that is not covered by annotations in .java files. If a value is missing or
# key is not provided, then the default value is assumed.
# This file is expected to be found in the root of an EJB project.

# Name of the schema file with DDL for export into external DBMS.
ddl.file=Tables.ddl

# Name of the backend that will be used by the mapping.
backend.id=DB2_v82

# Name of the database that will be used by the mapping.
database.name=PETSTORE

# Name of the schema that will be used by the mapping.
schema.name=DB2ADMIN
```

```
# Database vendor type that will be used by the mapping.
vendor.name=DB2UDBNT_V82

#vendor.name=SQL92
#vendor.name=SQL99
#vendor.name=DB2UDBNT_V61
#vendor.name=DB2UDBNT_V71
#vendor.name=DB2UDBOS390_V6
#vendor.name=DB2UDBAS400_V4
#vendor.name=ORACLE_V8
#vendor.name=INFORMIX_V92
#vendor.name=SYBASE_V1192
#vendor.name=SYBASE_V12
#vendor.name=MSSQLSERVER_V7
#vendor.name=MYSQL_V323
#vendor.name=INSTANTDB_V326
#vendor.name=INFORMIX_V73
#vendor.name=MSSQLSERVER_V70
#vendor.name=DB2UDBOS390_V7
#vendor.name=DB2UDBAS400_V5
#vendor.name=DB2UDBNT_V72
#vendor.name=CLOUDSCAPE_V50
#vendor.name=ORACLE_V9
#vendor.name=SYBASE_V125
#vendor.name=INFORMIX_V93
#vendor.name=DB2FAMILY
#vendor.name=DB2UDBNT_V8
#vendor.name=DB2EVERYPLACE_V81
#vendor.name=CLOUDSCAPE_V51
#vendor.name=INFORMIX_V94
#vendor.name=DB2UDBAS400_V52
#vendor.name=DB2UDBAS400_V53
#vendor.name=DB2UDBOS390_V8
#vendor.name=DB2CLOUDSCAPE_V82
#vendor.name=ORACLE_V10
#vendor.name=DB2EXPRESS_V81
#vendor.name=DB2EXPRESS_V82
```

---

**Tip:** The file `database.properties` in EJB project root directory is simply a collection of properties that rules the behavior for the WRD CMP mapping generation. This is closely related to the database types supported by the **ejbdeploy** command. More information is available at the following URL:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.etools.ejbbatchdeploy.doc/topics/regenc.html?resultof=%22%44%42%32%55%44%42%5f%56%38%32%22%20>

5. You are now ready to import the source code into the project. Select the **xPetstore-EJB** project, click **File** → **Import** → **File system** and click **Next**. Select the files from the xPetstore EJB application as shown in the following table and figure.

**Note:** To ease the identification of the source files to import, we included in Table 7-1 the directory structure where you can find each file.

Table 7-1 Import source files from xPetstore EJB application

Property	Value
From directory	<source_home>\xpetstore-ejb\java
Files to be imported	java\xpetstore\domain\customer\ejb\CustomerEJB.java java\xpetstore\domain\signon\ejb\AccountEJB.java java\xpetstore\services\petstore\ejb\PetstoreEJB.java java\xpetstore\services\petstore\exceptions\DuplicateEmailException.java java\xpetstore\services\petstore\exceptions\DuplicateAccountException.java java\xpetstore\util\ChainedException.java java\xpetstore\util\ChainedRuntimeException.java java\xpetstore\util\CreditCardNames.java java\xpetstore\util\Debug.java java\xpetstore\util\JMSUtil.java java\xpetstore\util\JNDINames.java java\xpetstore\util\Page.java
Into folder	xPetstore-EJB/ejbModule
Options	Create selected folders only

**Note:** We will be using the J2EE Perspective within Rational Application Developer V6 throughout the rest of this exercise.

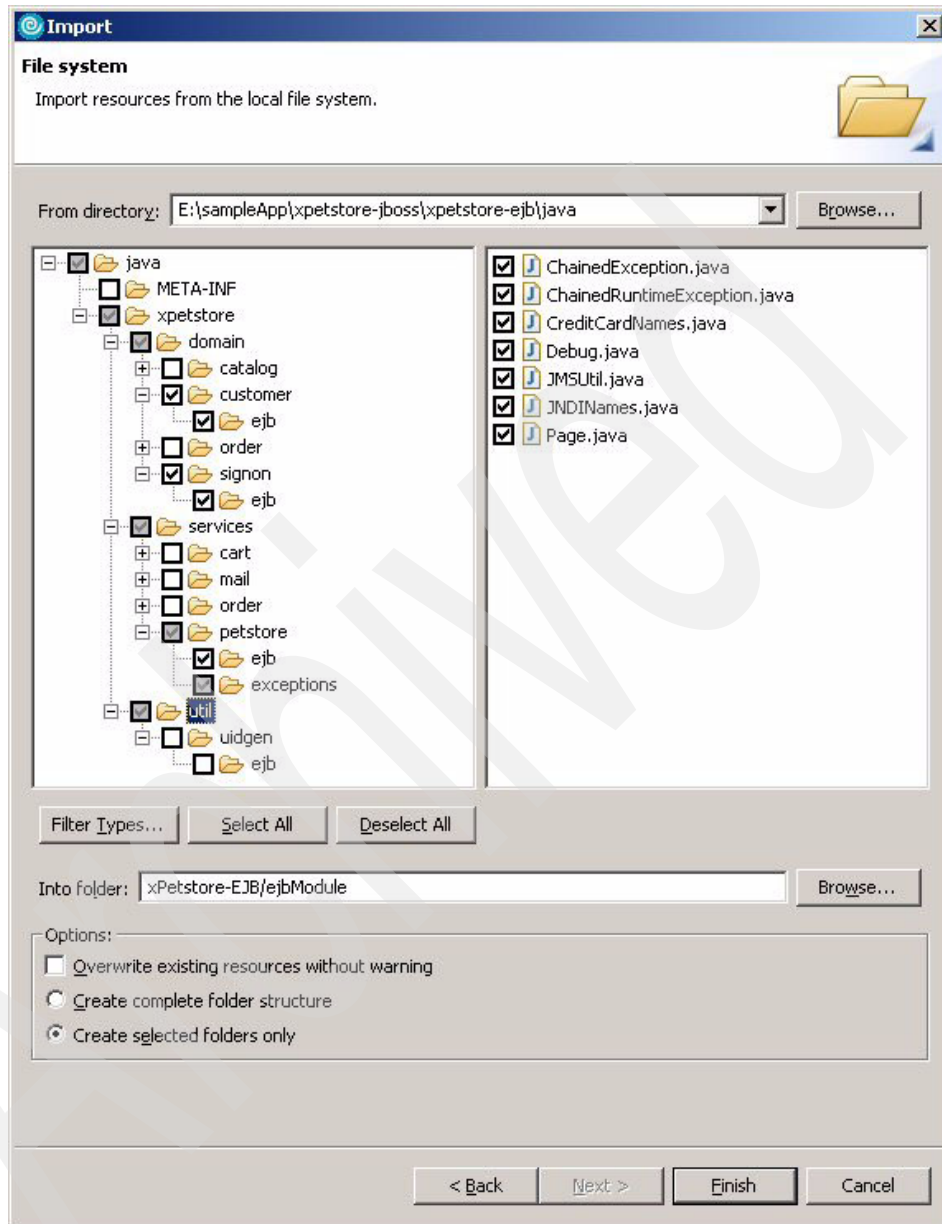


Figure 7-5 Import source files into the EJB project

Once you click **Finish** and the import is complete, you should see the directory structure as shown in Figure 7-6 on page 198.

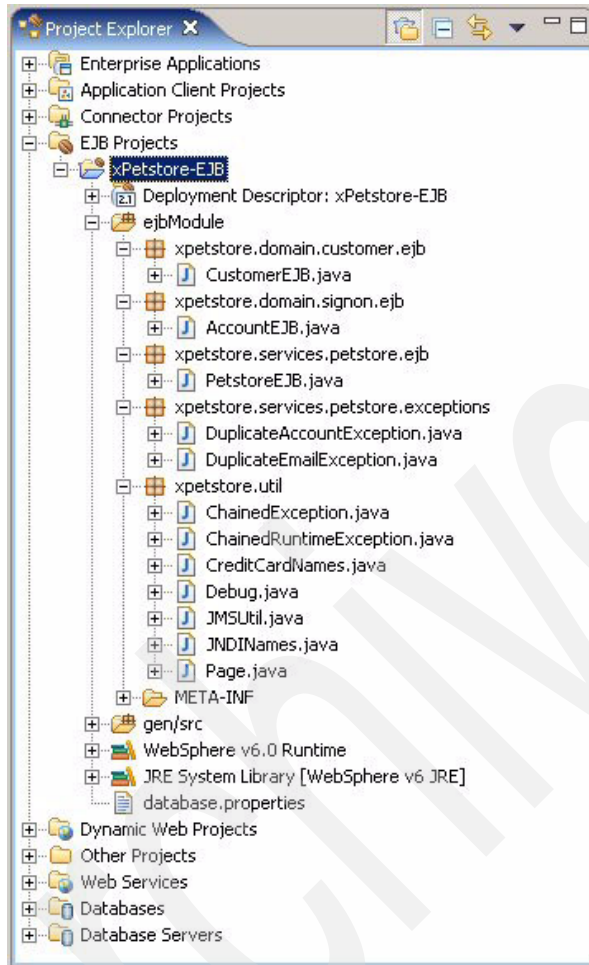


Figure 7-6 Files imported into the xPetstore-EJB project

- Click the **xPetstore-EJB** project and press **Ctrl+B** to rebuild the project. You will see a number of errors in the Problems view, as shown in the following figure.

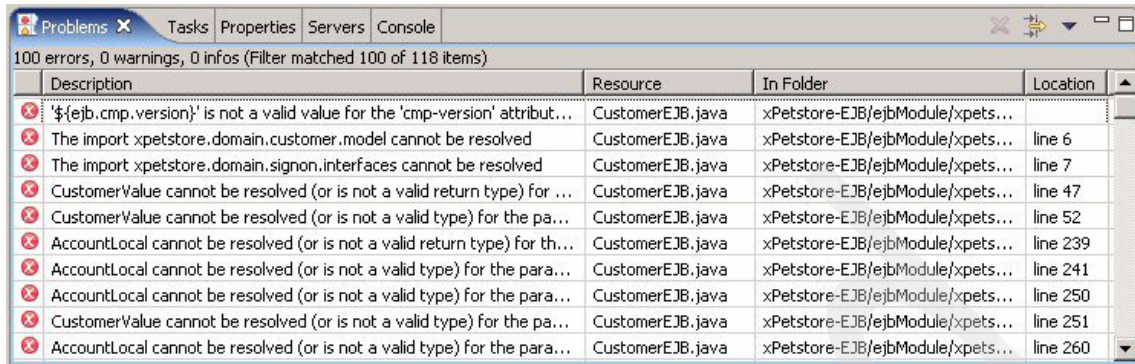


Figure 7-7 Problems after initial build of the imported files

7. From now on, we will choose to see only errors and warnings related to the resource selected instead of seeing all errors for all projects in the workspace. To reset the errors filter, you need to click the **Filters** icon in the top right side of the Problems view; in the dialog, select the option **On selected resource and its children** and click **OK**.
8. Open CustomerEJB.java file and inspect the code. Since the original application used XDoclet and Ant to build the project and generate artefacts such as the deployment descriptor and remote and home interfaces, you can see variable substitutions in XDoclet tags, such as `${ejb.cmp.version}` as shown in Example 7-8.

*Example 7-8 Original XDoclet tags in CustomerEJB.java file*

```

* @ejb.bean
*     name="Customer"
*     type="CMP"
*     view-type="local"
*     primkey-field="userId"
*     schema="Customer"
*     cmp-version="${ejb.cmp.version}"
* @ejb.value-object
*     name="Customer"
*     match="*"
* @ejb.transaction
*     type="Required"
* @ejb.persistence
*     table-name="T_CUSTOMER"
* @ejb.finder
*     signature="Customer findByEmail(java.lang.String email)"
*     query="SELECT OBJECT(c) FROM Customer AS c WHERE c.email = ?1"
*
* @jboss.persistence

```

```
*      create-table="${jboss.create.table}"
*      remove-table="${jboss.remove.table}"
*/
```

---

The variable substitution is not currently supported by WRD. As we perform this migration, we are not planning to maintain the original build environment using Ant and XDoclet. We will build the project with Rational Application Developer V6 and replace XDoclet with WRD.

**Tip:** You can have automated builds using headless Eclipse build as it is described in the IBM developerWorks® article “Performing unattended daily builds with WebSphere Studio and Ant” dated April 2004. This article also applies to Rational Application Developer V6:

[http://www.ibm.com/developerworks/websphere/library/techarticles/0404\\_bowker/0404\\_bowker1.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0404_bowker/0404_bowker1.html)

To remove compilation problems, you will have to edit all the imported files and replace variable substitution used in XDoclet tags with concrete values as we describe in the following steps.

Note that at this point, WRD interprets all XDoclet tags that we have in the source code. WRD was modeled after XDoclet and partial tags compatibility was one of the design goals.

9. Delete the string “`${ejb.cmp.version}`” next to the `cmp-version=`, then position your cursor right after the `=` sign and press **Ctrl+”Space”**. This will result in automatic code assistance; since there are no other choices, your line will be completed with the “`2.x`” value.
10. Look through the file and make sure to delete all tags related to JBoss and WebLogic, such as `@jboss.*` and `@weblogic.*` and others. You can run a search to facilitate the task.
11. Press **Ctrl+B** to rebuild the project; you should see some of the errors disappear.
12. Look at the Problems view. The first error should be `@ejb.pk-field` without a valid `@ejb.pk` tag on the enclosing class. Fix this error by adding the `@ejb.pk class="java.lang.String"` tag to the class-wide declaration right after the `@ejb.finder` tag. See Example 7-9 on page 201 for the fully corrected version.
13. The next error to fix is `CustomerValue` cannot be resolved (or is not a valid return type) for the method `getCustomerValue`. This is a so-called Value Object, an implementation of the Data Transfer Object pattern.



**Note:** The order in which we fix the errors may not necessarily be the same as shown in the Problems view.

XDoclet automatically generates the name with the Value at the end of it. WRD needs to be told the exact class name. Fix this by editing the class-wide WRD tag to make it look like the following example:

```
* @ejb.value-object
*     name="CustomerValue"
*     match="*"
```

Here is the fully corrected version of the class-wide WRD tag.

*Example 7-9 Corrected WRD tags in CustomerEJB.java file*

---

```
... *
* @ejb.bean
*     name="Customer"
*     type="CMP"
*     view-type="local"
*     primkey-field="userId"
*     schema="Customer"
*     cmp-version="2.x"
* @ejb.value-object
*     name="CustomerValue"
*     match="*"
* @ejb.transaction
*     type="Required"
* @ejb.persistence
*     table-name="T_CUSTOMER"
* @ejb.finder
*     signature="Customer findByEmail(java.lang.String email)"
*     query="SELECT OBJECT(c) FROM Customer AS c WHERE c.email = ?1"
* @ejb.pk class="java.lang.String"
*
*/
```

---

14. After rebuilding the project, there will be a few errors remaining for this EJB. One of them is shown next:

The import `xpetstore.domain.customer.model` cannot be resolved -  
CustomerEJB.java - xPetstore-EJB/ejbModule/xpetstore/domain/customer/ejb -  
line 6

This happens because WRD always generates value objects in the same package as the EJB class while XDoclet can be configured to place a value object in the package of your choice. In our case, we need to delete the import statement or let the tool fix all imports automatically. Anywhere in the

Java text of the file CustomerEJB.java, right-click and select **Source** → **Organize Imports**. This will fix all import statements automatically.

15. Once you rebuild the project, pressing **Ctrl+B** will fix cause most of the errors to disappear. We will deal with the remaining errors as we fix the Account bean.
16. You will need to perform similar changes to the AccountEJB.java file. Repeat the previous steps on AccountEJB.java to correct the appropriate tags in this file. Example 7-10 shows the class-wide WRD tag already corrected.

*Example 7-10 Updated WRD tags in the AccountEJB.java file*

---

```
...
* @ejb.bean
*     name="Account"
*     type="CMP"
*     view-type="local"
*     primkey-field="userId"
*     schema="Account"
*     cmp-version="2.x"
* @ejb.value-object
*     name="AccountValue"
*     match="*"
* @ejb.transaction
*     type="Required"
* @ejb.persistence
*     table-name="T_ACCOUNT"
* @ejb.pk class="java.lang.String"
*
*/
```

---

17. Once you have rebuilt the project, you will still see some compilation errors for the CustomerEJB. This is because of the import statements. As with value objects, in XDoclet you can redirect home interfaces and other generated files into different packages, but WRD will use the same package for generated artefacts as the class that is annotated. Go back to the CustomerEJB.java file and automatically fix all imports (press **Ctrl+Shift+O**), then rebuild the project. The errors for CustomerEJB and AccountEJB should have disappeared.

Now, inspect the artefacts that were generated. Take a look at local interfaces for the beans. All the code that was generated by WRD can be found in the xPetstore-EJB/gen/src folder.

For example, take a look at the CustomerLocal.java file within the xpetstore.domain.customer.ejb package, which is the local interface for the EJB. This looks exactly like the file that was generated by XDoclet and it can be found in the following directory:

```
<source_home>\xpetstore-ejb\build\java\xpetstore\domain\customer\interfaces  
\CustomerLocalHome.java
```

*Example 7-11 CustomerLocal.java file content*

```
package xpetstore.domain.customer.ejb;  
public interface CustomerLocal extends javax.ejb.EJBLocalObject {  
    public xpetstore.domain.customer.ejb.CustomerValue getCustomerValue();  
    public void setCustomerValue(xpetstore.domain.customer.ejb.CustomerValue  
data);  
    public java.lang.String getUserId();  
    public xpetstore.domain.signon.ejb.AccountLocal getAccount();  
}
```

18. Navigate to the file `CustomerLocalHome.java` located in the `xPetstoreEJB/gen/src/xpetstore.domain.customer.ejb` folder. If you look at the Problems view, you will see a warning:

```
CHKJ2403W: This method must return CustomerLocal or a collection thereof  
(EJB 2.0: 9.6.2, 10.5.6, 10.6.12, 11.2.2).CustomerLocalHome.java  
xPetstore-EJB/gen/src/xpetstore/domain/customer/ejbline 24
```

This is because of the difference in how XDoclet and WRD generate finder methods from the tag shown below:

```
/**  
 * @ejb.finder  
 * signature="Customer findByEmail(java.lang.String email)"  
 * query="SELECT OBJECT(c) FROM Customer AS c WHERE c.email = ?1"  
 */
```

For WRD to generate the proper finder method signature, you need to change `Customer` to `CustomerLocal` in the `CustomerEJB.java` file as shown below:

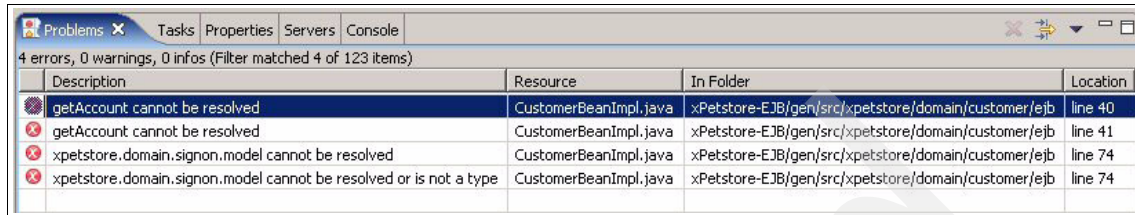
```
/**  
 * @ejb.finder  
 * signature="CustomerLocal findByEmail(java.lang.String email)"  
 * query="SELECT OBJECT(c) FROM Customer AS c WHERE c.email = ?1"  
 */
```

Now, when you rebuild the project, you will see that the finder method on the `CustomerLocalHome.java` file looks correct and shows no errors:

```
public interface CustomerLocalHome extends javax.ejb.EJBLocalHome {  
    . . .  
    CustomerLocal findByEmail(java.lang.String email)  
        throws javax.ejb.FinderException;  
}
```

19. Now take a look at the errors for the file `CustomerBeanImpl.java`. This was generated by WRD and is configured in the `ejb-jar.xml` as our bean

implementation class for the Customer EJB. There are some errors in this file, as illustrated in Figure 7-8 on page 204.



Description	Resource	In Folder	Location
getAccount cannot be resolved	CustomerBeanImpl.java	xPetstore-EJB/gen/src/xpetstore/domain/customer/ejb	line 40
getAccount cannot be resolved	CustomerBeanImpl.java	xPetstore-EJB/gen/src/xpetstore/domain/customer/ejb	line 41
xpetstore.domain.signon.model cannot be resolved	CustomerBeanImpl.java	xPetstore-EJB/gen/src/xpetstore/domain/customer/ejb	line 74
xpetstore.domain.signon.model cannot be resolved or is not a type	CustomerBeanImpl.java	xPetstore-EJB/gen/src/xpetstore/domain/customer/ejb	line 74

Figure 7-8 CustomerBeanImpl.java compilation errors

These errors are because WRD does not generate value objects for Container Managed Relationship (CMR) between Customer and Account EJBs.

To work around the problem, remove the definition of the value object for CMR in the CustomerEJB.java file. Simply remove the `@ejb.value-object` tag and leave the definition of CMR as shown in Example 7-12.

Example 7-12 CustomerEJB.java CMR definition removed

```
...
/**
 * @ejb.interface-method
 * @ejb.relation
 *     name="customer-account"
 *     role-name="customer-has-account"
 *     target-ejb="Account"
 *     target-role-name="account-belongs_to-customer"
 *     target-cascade-delete="yes"
 */
// FIXME - WRD value objects with CMRs do not work at this time
//     * @ejb.value-object
//     * compose="xpetstore.domain.signon.model.AccountValue"
//     * compose-name="AccountValue"
//     * members="xpetstore.domain.signon.interfaces.Account"
//     * members-name="AccountValue"
//     * relation="external"
public abstract AccountLocal getAccount();
...
```

20. Add `setAccount()` to the remote interface by adding the `@ejb.interface-method` tag as highlighted in the following example:

```
/**
 * @ejb.interface-method
 */
```

```
public abstract void setAccount(AccountLocal account);
```

21. As we save and rebuild the project, all errors for the CustomerBeanImpl.java file disappear, but we still see a couple of errors in the CustomerValue.java file.

```
pk.userId cannot be resolved or is not a field
    CustomerValue.java
    xPetstore-EJB/gen/src/xpetstore/domain/customer/ejb
    line 154
```

```
The method setUserId(String) is undefined for the type String
    CustomerValue.java
    xPetstore-EJB/gen/src/xpetstore/domain/customer/ejb
    line 162
```

This is because there is an extra tag in the CustomerEJB.java file that confuses WRD:

```
/**
 * @ejb.pk-field
 * . . .
 */
public abstract String getUserId( );
```

If you remember, we have already defined the primary key for our bean at the class level tag, as shown below:

```
/**
 * @ejb.bean
 *     name="Customer"
 *     type="CMP"
 *     view-type="local"
 *     primkey-field="userId"
 *     . . .
 */
```

Therefore, we can simply remove the @ejb.pk-field tag from the CustomerEJB.java file and rebuild the project. Now none of the files in the package xpetstore.domain.customer.ejb have any errors.

You may still have a Customer.java file if it was not automatically removed when you changed the @ejb.value-object from Customer to CustomerValue. If you still have this file, it will continue to have errors; feel free to remove this file since this is an old value object that you no longer need.

22. After you save and rebuild the project, you should still see a couple of errors in the AccountValue.java file.

```
pk.userId cannot be resolved or is not a field
    AccountValue.java
    xPetstore-EJB/gen/src/xpetstore/domain/signon/ejb
    line 43
```

The method `setUserId(String)` is undefined for the type `String`  
AccountValue.java  
xPetstore-EJB/gen/src/xpetstore/domain/signon/ejb  
line 51

This is because there is extra tag in the AccountEJB.java file that confuses WRD:

```
/**
 * @ejb.pk-field
 * . . .
 */
public abstract String getUserId( );
```

If you remember, we have already defined the primary key for our bean at the class level tag, as shown below:

```
/**
 * @ejb.bean
 *     name="Customer"
 *     type="CMP"
 *     view-type="local"
 *     primkey-field="userId"
 * . . .
 */
```

Therefore, we can simply remove the `@ejb.pk-field` tag from the AccountEJB.java file and rebuild the project. Now none of the files in the package `xpetstore.domain.signon.ejb` have any errors.

23. When we performed our migration, we found that there were transaction deadlocks because of the way the code accesses Account and Customer EJB beans during the new EJB creation. To solve these deadlocks, we changed the `ejbCreate()` method in the CustomerEJB.java file as highlighted in the following example:

```
/**
 * @ejb.create-method
 */
public String ejbCreate( AccountLocal account, CustomerValue data )
    throws CreateException
{
    // setUserId(account.getUserId());
    setUserId(data.getUserId());
    setCustomerValue(data);
    return null;
}
```

24. Take a look at the Session bean PetstoreEJB.java. There are compilation errors in that file. The first error is:

The '@ejb.transaction' tag may only occur in a single 'method' 0..1 times.  
This is occurrence number '2'.

```
PetstoreEJB.java  
xPetstore-EJB/ejbModule/xpetstore/services/petstore/ejb
```

This is easy to fix; simply double-click the error message in the Problems view. This will take you to the file where the error is. As you can see, the transaction tag is repeated twice. Simply remove the duplicated entry, so that you only have that tag defined once.

25. Another compilation error in the PetstoreEJB.java file is the reference to CustomerUtil and AccountUtil classes. XDoclet generates these by default, but WRD needs to be told to generate these. Simply add the following tag to the beginning of a class-wide declaration for PetstoreEJB.java, CustomerEJB.java and AccountEJB.java files, somewhere after the @ejb.bean tag, as highlighted in the following example:

```
/**  
 * @ejb.bean name= . . .  
 * . . .  
 * @ejb.util generate="physical"  
 */
```

This will force WRD to generate helper classes for JNDI lookup and obtaining references to the Home interfaces for the beans where this tag is defined. It will generate files with the pattern <bean\_name>Util.java.

26. The next set of errors in the PetstoreEJB.java file is comprised of import statements. Since we did not perform a complete migration, we did not import all class definitions for the application. For simplicity, we will remove all of the imports that are broken and correct Customer and Account bean related imports by changing the package names.

Anywhere in the PetstoreEJB.java file, right-click and select **Source** → **Organize Imports**. Now save the file and rebuild the project. This will fix import errors.

```
import xpetstore.domain.customer.ejb.CustomerLocal;  
import xpetstore.domain.customer.ejb.CustomerLocalHome;  
import xpetstore.domain.customer.ejb.CustomerUtil;  
import xpetstore.domain.customer.ejb.CustomerValue;  
import xpetstore.domain.signon.ejb.AccountLocal;  
import xpetstore.domain.signon.ejb.AccountLocalHome;  
import xpetstore.domain.signon.ejb.AccountUtil;  
import xpetstore.domain.signon.ejb.AccountValue;
```

27. There are still errors in the PetstoreEJB.java file:

```
Referenced bean 'Category' does not exist...  
Referenced bean 'Item' does not exist...  
Referenced bean 'Order' does not exist...  
Referenced bean 'Product' does not exist...
```

This is because we do not have those EJBs in our project. Simply remove @ejb.ejb-ref WRD tags related to those beans.

28. Remove all JBoss and WebLogic related XDoclet tags.

29. There is a difference between XDoclet and WRD in how they generate JNDI names in <Bean>LocalHome class. In the case of a Customer bean, XDoclet generates JNDI names as follows:

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

While WRD generates them as follows:

```
public interface CustomerLocalHome extends javax.ejb.EJBLocalHome {
    public static final String COMP_NAME="java:comp/env/ejb/Customer";
    public static final String JNDI_NAME=
"ejb/xpetstore/domain/customer/ejb/CustomerLocalHome";
```

Note the difference in using the <Bean>Local suffix in XDoclet. This is not of tremendous importance, but means that you have to edit the EJB references in the PetstoreEJB.java file as shown in Example 7-13. We simply removed the Local from the end of ref-name attribute.

*Example 7-13 PetstoreEJB.java corrected for WRD*

---

```
/**
 * . . .
 * @ejb.ejb-ref
 *     ejb-name="Customer"
 *     view-type="local"
 *     ref-name="ejb/Customer"
 * @ejb.ejb-ref
 *     ejb-name="Account"
 *     view-type="local"
 *     ref-name="ejb/Account"
 * . . .
```

---

30. Since we do not cover migration of the MDB in this example, we can also remove all @ejb.resource-ref tags from the PetstoreEJB.java file. This tag is supported by WRD, but we simply do not need those resource references.

31. Save the file and rebuild the project.

32. There are still a number of compilation errors left; this is because of the methods in PetstoreEJB.java that use classes we left outside this project. Remove all these methods from the file (you may want to use the Outline view to ease the removal of these methods):

```
getCategory(String)
getCategories(int, int)
```



```

getProduct(String)
getProductByItem(String)
getProducts(String, int, int)
searchProducts(String, int, int)
getItem(String)
getItems(String, int, int)
getCategoryLocalHome()
getProductLocalHome()
getItemLocalHome()
createOrder(String, Date, Map)
getCustomerOrders(String, int, int)
getOrder(Integer)
getOrderItems(Integer, int, int)
getOrderLocalHome()
toPage(Collection, int, int, Class)
getData(Object, Class)

```

33. In a real migration, you will likely comment out those methods and return to them later as you get one or two beans working properly. Performing a migration in small incremental steps is a good idea. Here are methods that we still have in the `PetstoreEJB.java` file; all other methods were removed. Anywhere in the `PetstoreEJB.java` file, press **Ctrl+O** to see a quick outline of the class.

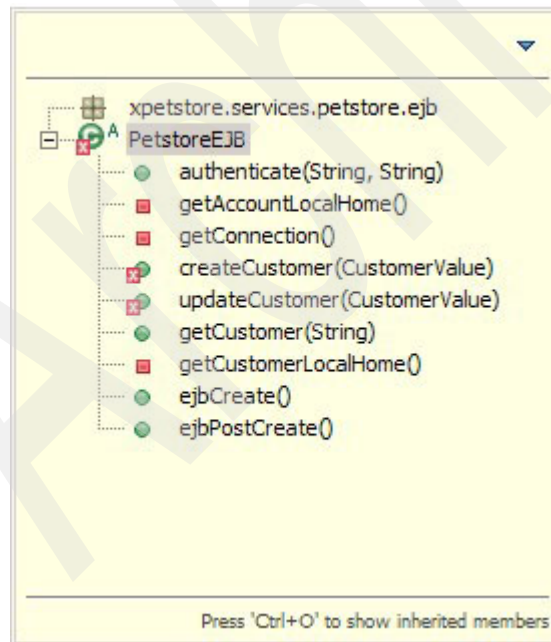


Figure 7-9 Outline of the `PetstoreEJB.java` file

34. Since you removed the value object tag from the CMR definition in CustomerEJB.java file, somehow you need to pass account information outside of the CustomerValue class.

Since the account information is no longer part of the CustomerValue object, you need to add AccountValue to the createCustomer() method signature in the PetstoreEJB.java file:

```
public String createCustomer(CustomerValue customer, AccountValue account)
throws DuplicateAccountException, DuplicateEmailException, CreateException
{ . . .
```

35. You also need to change the body of the createCustomer() method to handle the new parameters properly. Example 7-14 shows the final version of this method.

*Example 7-14 createCustomer method in PetstoreEJB.java file*

---

```
/**
 * @ejb.interface-method
 */
public String createCustomer(CustomerValue customer, AccountValue account) throws
DuplicateAccountException, DuplicateEmailException, CreateException
{
    /* Make sure that the customer ID is unique */
    CustomerLocalHome home = getCustomerLocalHome();
    try {
        String id = customer.getUserId();
        home.findByPrimaryKey(id);
        throw new DuplicateKeyException(id);
    } catch (FinderException f) {
        AccountLocal act = null;
        try {
            act = getAccountLocalHome().create(account);
        } catch (DuplicateKeyException d) {
            throw new DuplicateAccountException(account.getUserId(), d);
        }
        CustomerLocal cst = home.create(act, customer);
        return cst.getUserId();
    }
}
```

---

36. You also need to make similar changes to the updateCustomer() method. Example 7-15 shows the final version of this method.

*Example 7-15 updateCustomer method in PetstoreEJB.java file*

---

```
/**
 * @ejb.interface-method
 */
```

```

public void updateCustomer(CustomerValue customer, AccountValue account) throws
DuplicateEmailException, FinderException
{
    /* Make sure that the email is unique */
    CustomerLocalHome home = getCustomerLocalHome();
    CustomerLocal cst = null;
    /* Update the customer */
    if (cst == null) {
        cst = home.findByPrimaryKey(customer.getUserId());
    }
    cst.setCustomerValue(customer);

    if ((account!=null)&&(account.getPassword()!=null)&&(account.getPassword().length()>0))
    {
        cst.getAccount().setAccountValue(account);
    }
}

```

---

37. And finally, because CustomerValue does not include account information, you need to add a new method to the PetstoreEJB.java file to access account information, as shown in Example 7-16.

*Example 7-16 New method getCustomerAccount() in PetstoreEJB.java file*

```

/**
 * @ejb.interface-method
 */
public AccountValue getCustomerAccount(String userId) throws FinderException {
    return getCustomerLocalHome().findByPrimaryKey(userId).getAccount().getAccountValue();
}

```

---

38. Save the changes and rebuild the project.

39. You should still see some warnings in the Problems view. To fix these warnings, right-click anywhere within the PetstoreEJB.java file and select **Source** → **Organize Imports**, then save the changes and rebuild the project.

You should see no errors in the xPetstore-EJB project. You have finished the source code migration from XDoclet to WRD tags, but there are a few more things you need to do before you can run the application.

**Note:** You may still see some database mapping errors, for example, one stating that CustomerEJB is not mapped to a table. During the previous rebuild, Rational may not have mapped all the beans. This error is fixed by just making any update to the CustomerEJB.java so Rational will recognize a change in the source and will allow you to perform another build. This new build should fix all the errors in the xPetstore-EJB project.

## Verifying database mappings for CMPs

Now that we have created our CMPs, the next step would be to define the back-end database. The good news is that we have created a `database.properties` file already and our database mappings were generated automatically for the back end specified in our properties file.

The source code from the original application includes EJB to DBMS mappings defined as WRD tags (originally XDoclet tags) inside of the respective `*EJB.java` files. Therefore, you do not need to manually do the database mapping by using wizards to create a back-end folder. WRD automatically generates mappings in the `xPetstore-EJB/ejbModule/META-INF/backends/DB2_v82` folder.

Double-click the `map.mapxmi` file mapping editor and expand the Enterprise Beans and Tables; you should see a window similar to the one shown in Figure 7-10.

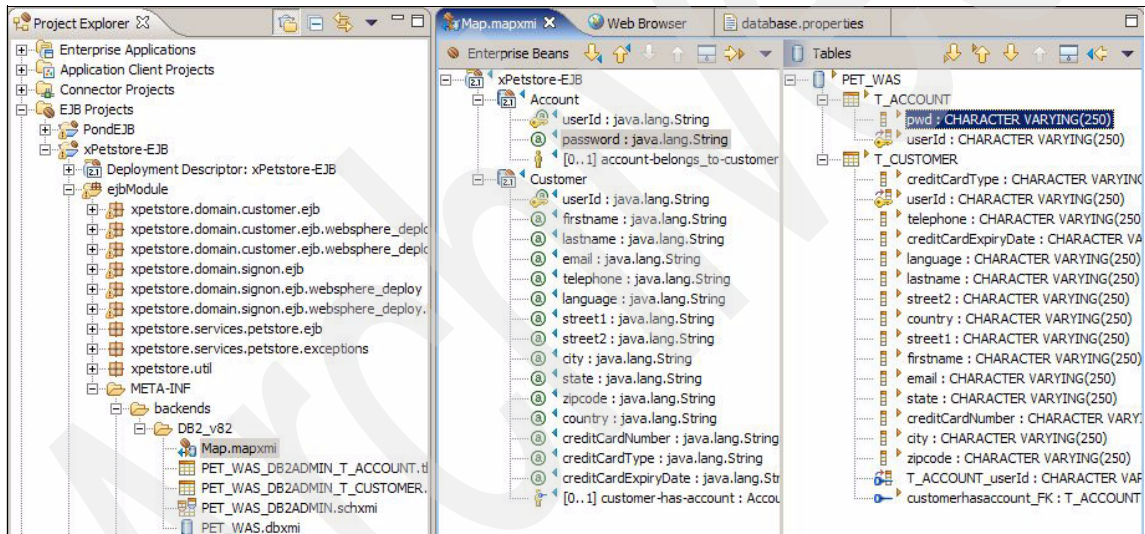


Figure 7-10 EJB to RDBMS mapping editor

To check that an object has not been mapped, click the **Show only the unmapped objects** icon in the mapping editor. Since all fields are mapped, you should see an empty list under xPetstore-EJB project in the Enterprise Beans window. Optionally, you can right-click the table name and open it in the editor where you can change the name and the definition of the table, including column names and types.

**Restriction:** The current implementation of WRD ignores SQL types specified in the WRD tag and generates default mappings. For example, in CustomerEJB.java, we have lastname property defined as follows:

```
/**
 * @ejb.persistence
 *     column-name="lastname"
 *     jdbc-type="VARCHAR"
 *     sql-type="varchar(50)"
 */
```

Notice that the type of the LASTNAME column (in the Tables window on the right) is incorrectly generated as CAHARCTER VARYING(250). This is also the case for all String properties of the Customer bean.

On the table side (to the right side of the window), all database columns are mapped to respective EJB properties, except for the T\_ACCOUNT\_userId column, which is our CMR column. This is because the foreign key on the T\_ACCOUNT table is mapped to the EJB, but not the column itself. The name of this column as it is generated by default is different from what was generated by the JBoss when it automatically created tables in the database. You have two options to deal with this:

- You can add the @websphere-cmr.column tag to the CustomerEJB.java file where you defined the CMR and specify the name of the column to be the same as it was generated by JBoss - ACCOUNT\_FK:

```
/**
 * . . .
 * @websphere-cmr.column
 *     name="account_fk"
 *     jdbc-type="VARCHAR"
 *     sql-type="varchar(10)"
 */
public abstract AccountLocal getAccount();
```

This way, you can map the bean to the existing database schema that was generated by JBoss. Unfortunately, due to a problem with WRD, the generated database column is of type BIT VARYING(1) instead of the proper VARCHAR(10).

You could manually change the definition of the table, but you will have to do it each time you rebuild the back-end mappings since your table definition will be overridden.

- Alternatively, you can use the database schema that is generated by WRD, export the DDL file and create new tables.

In a real migration, you probably would not have the luxury of creating your own table definitions, but for this migration example, we decided to use this method to work around the error discussed above.

The following steps will guide you in creating and modifying a DDL file in order to have the proper table names and types in the database.

1. In the J2EE perspective of Rational Application Developer V6, right-click the **xPetstore-EJB** project and select **EJB to RDB mapping** → **Generate Schema DDL**. This will create the file **Table.ddl** in the **xPetstore-EJB/ejbModule/META-INF/backends/DB2\_v82** folder.
2. Double-click the **Table.ddl** file to open it in text editor. Add lines as shown next in **bold** to the file and change all values of 250 to 100.

*Example 7-17 Table.ddl update*

---

```
-- These two lines go to the very top of the file
drop TABLE DB2ADMIN.T_ACCOUNT;
drop TABLE DB2ADMIN.T_CUSTOMER;

-- Generated by Relational Schema Center for SQL-92
-- *GENERATED*

CREATE SCHEMA DB2ADMIN;

-- *GENERATED*

CREATE TABLE DB2ADMIN.T_ACCOUNT
  (pwd CHARACTER VARYING(100),
   userId CHARACTER VARYING(100) NOT NULL);

ALTER TABLE DB2ADMIN.T_ACCOUNT
  ADD CONSTRAINT PK_T_ACCOUNT PRIMARY KEY (userId);

-- *GENERATED*

CREATE TABLE DB2ADMIN.T_CUSTOMER
  (creditCardType CHARACTER VARYING(100),
   userId CHARACTER VARYING(100) NOT NULL,
   telephone CHARACTER VARYING(100),
   creditCardExpiryDate CHARACTER VARYING(100),
   language CHARACTER VARYING(100),
   lastname CHARACTER VARYING(100),
   street2 CHARACTER VARYING(100),
   country CHARACTER VARYING(100),
   street1 CHARACTER VARYING(100),
   firstname CHARACTER VARYING(100),
   email CHARACTER VARYING(100),
   state CHARACTER VARYING(100),
```

```

        creditCardNumber CHARACTER VARYING(100),
        city CHARACTER VARYING(100),
        zipcode CHARACTER VARYING(100),
        T_ACCOUNT_userId CHARACTER VARYING(100));

ALTER TABLE DB2ADMIN.T_CUSTOMER
    ADD CONSTRAINT PK_T_CUSTOMER PRIMARY KEY (userId);

-- This line goes to the very bottom of the file
INSERT INTO DB2ADMIN.T_ACCOUNT(userid, pwd) VALUES ('user1', 'password1');

```

---

**Note:** The problem of the wrong SQL type generation is a known one and will be addressed in the next Rational Application Developer V6 fixpack.

3. Save the file and export it anywhere into the file system, for example, save it into the directory C:\temp.

4. From a command line window, start the DB2 command line processor:

```

cd c:\temp
db2cmd

```

5. Now, while you are in the DB2 command prompt, execute the following commands to replace the T\_CUSTOMER and T\_ACCOUNT tables in the existing PETSTORE database with new definitions using the Table.ddl file:

```

db2 connect to petstore user db2admin using its0ra10
db2 -tvf Table.ddl

```

At this point, the database is ready to handle persistence for the Customer and Account CMP beans.

## Migrating the test Web project

To test the EJB project you migrated in the previous section, you will need to import and run JUnitEE Web application that is provided with xPetstore EJB. JUnitEE is an extension to JUnit and provides servlet-based reporting in addition to command line, Ant and GUI interfaces of JUnit.

1. Create a new Web project and add it to your xPetstore-EAR application. Click **File** → **New** → **Dynamic Web Project**. Create a new Dynamic Web Project with the following characteristics and click **Next**.

Table 7-2 New dynamic web project dialog

Parameter	Value
Name	xPetstore-junit-WEB
EAR Project	xPetstore-EAR (Note: this EAR was generated when we created xPetstore-EJB project)
Context Root	junitee_test
Add support for annotated Java classes	Checked

- In the next window, make sure you select **xPetstore-EJB.jar** and then click **Finish**.
- Now you need to use some files that were generated by XDoclet JUnitEE plugin during the Ant build for the original application in the JBoss environment.  
  
Unzip the index.html file from the <source\_home>\dist\xpetstore-ejb-test.war file.
- In Rational Application Developer V6, select **xPetstore-junit-WEB** → **WebContent** folder, right-click it and select **Import**. In the dialog, select **File System** and click **Next**.
- In the next window, select <source\_home>\dist for the directory and check the index.html file for import. Click **Finish** and make sure that the file was imported right into the WebContent directory of the Web application you just created.  
  
This file was generated as part of the Ant build by the JUnitEE task. If you add more test cases, you may change the file manually or rerun the Ant build that will invoke JUnitEE and rebuild the content of the file based on the test cases it finds.
- You also need to import test cases and servlet that invokes those test cases. Right-click the folder **xPetstore-junit-WEB** → **Java Resources** → **JavaSource** and select **Import**. In the dialog window, select **File System** and click **Next**.
- Click the **Browse** button and navigate to the directory <source\_home>\xpetstore-ejb\test and click **OK**. Now expand the directory tree and select the **petstore** and **test** directories as shown in the following figure. Make sure that your Into folder is xPetstore-junit-WEB/JavaSource. Click **Finish**.



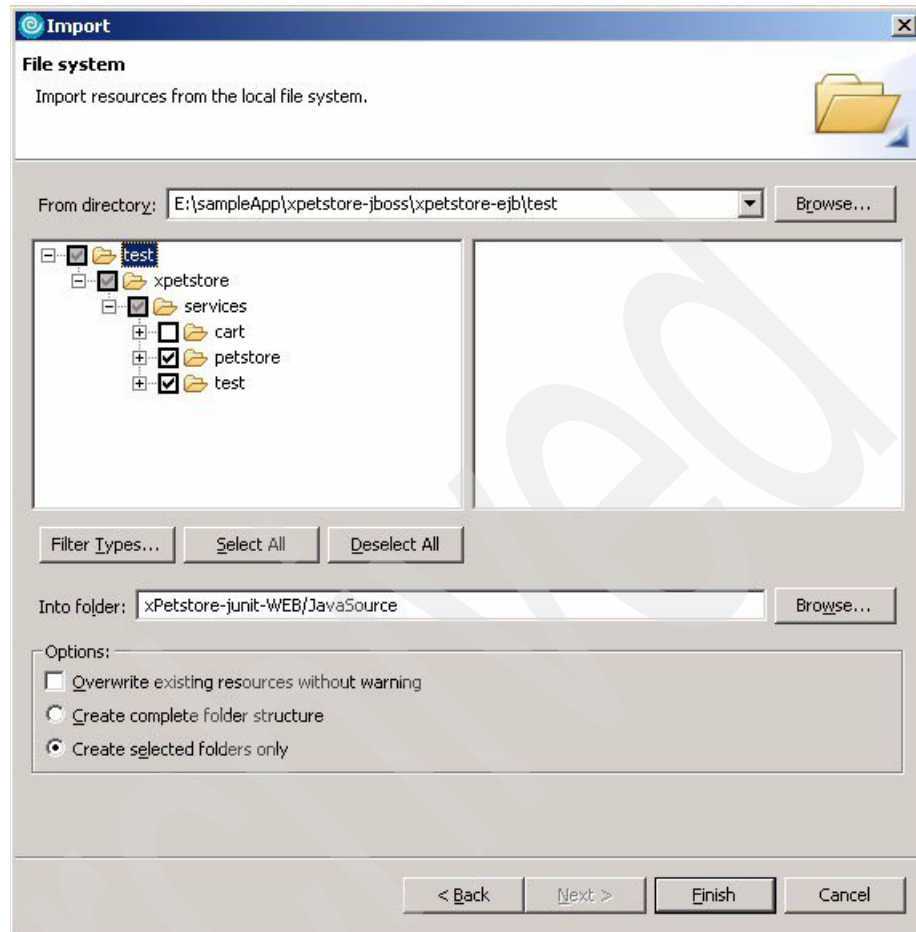


Figure 7-11 Import JUnitEE files into the project

8. As you rebuild the Web project, you will see compilation errors for the EJBTestServlet.java file, such as:

```
The import junit cannot be resolved                                PetstoreTest.java
xPetstore-junit-WEB/JavaSource/xpetstore/services/petstore/testline 9
```

Fix this error by importing the JUnitEE.jar file into the Web application. Select xPetstore-junit-WEB project and click **File** → **Import** → **File System** and click **Next**.

9. Click **Browse** and navigate to the folder <source\_home>\lib\main\org.junitee then click **OK**.
10. Check only the junitee.jar file.

11. Click the **Browse** button next to the **Into Folder**. Select the folder **xPetstore-junit-WEB/WebContent/WEB-INF/lib** and click **OK** and then **Finish** to complete the import operation.  
  
When you import jar files into the WEB-INF/lib directory of the Web project, they are added to the compile and runtime classpath of the Web application. This is part of the J2EE standard packaging scheme.
12. Repeat the import operation described above, but this time for the `<source_home>\lib\ant\junit\junit.jar` file.
13. Open the `EJBTestServlet.java` file in the Java editor and remove the tag `@web.ejb-local-ref` for the `ejb/CartLocal` reference since you do not have it in the EJB project.
14. For the same reason described in `PetstoreEJB.java` migration steps (the difference in JNDI names generated by WRD and XDoclet), you need to edit the EJB reference tag in the `EJBTestServlet.java` file so it looks like the following example (we replaced `ejb/PetstoreLocal` with `ejb/Petstore`).

*Example 7-18 EJBTestServlet.java - updated EJB reference tag*

---

```
/**
 * . . .
 * @web.ejb-local-ref
 *     name="ejb/Petstore"
 *     type="Session"
 *     home="xpetstore.services.petstore.interfaces.PetstoreLocalHome"
 *     local="xpetstore.services.petstore.interfaces.PetstoreLocal"
 *     link="Petstore"
 */
```

---

If you rebuild the `xPetstore-junit-WEB` project now, you will see that all errors for the `EJBTestServlet.java` are gone, but there are still errors for the `PetstoreTest.java` file.

15. Open the `PetstoreTest.java` file and press **Ctrl+Shift+O** to organize the imports and update import statements for the file.  
  
You also need to delete all methods that are related to beans that are not included into this migration. Here is the list of methods we have left in the file after editing.

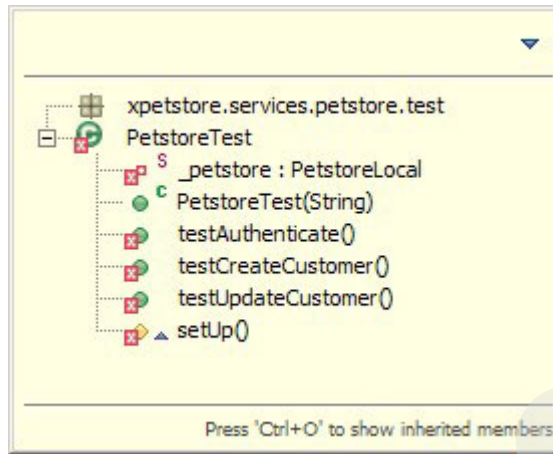


Figure 7-12 Methods in the PetstoreTest.java file

16. Since you changed the method signatures in the Petstore bean, you need to change some code in PetstoreTest class to handle these changes. The following example shows the updated methods.

*Example 7-19 testCreateCustomer() method in PetstoreTest class*

```
//=====
// Customer Test cases
//=====
public void testCreateCustomer() {
    try
    {
        String userId = "CST-" + (System.currentTimeMillis() % 100000);
        AccountValue act0 = new AccountValue(userId, "password");
        // Work around the fact that CustomerValue does not include account information
        // CustomerValue cst0 = new CustomerValue( null, "firstname", "lastname", userId . . .
        // cst0.setAccountValue( act0 );
        CustomerValue cst0 = new CustomerValue(userId, "firstname", "lastname", userId +
            "@foo.com", "123-456", "en", "street1.1", "street1.2", "city", "CA", "A1A-1A1", "US",
            "111-111-111", "Visa", "01-11");
        System.out.println("+++++++ creating customer=" + cst0);
        // Change to accomodate method signature
        // String id = _petstore.createCustomer( cst0 );
        String id = _petstore.createCustomer(cst0, act0);
        CustomerValue cst = _petstore.getCustomer(id);
        // Instead of getting account information from CustomerValue class we call new method
        // AccountValue act = cst.getAccountValue( );
        AccountValue act = _petstore.getCustomerAccount(id);

        // !!!!!!!!!!!!!!! THE REST OF THE METHOD GOES HERE WHITHOUT CHANGE !!!!!!!!!!!!!!!
    }
}
```

```

    assertEquals("firstname", "firstname", cst.getFirstname());
    . . .
}

```

17. You also need to make similar changes to the `testUpdateCustomer()` method as illustrated in the following example.

*Example 7-20 testUpdateCustomer() method in PetStoreTest class*

```

public void testUpdateCustomer() {
    try {
        String userId = "CST-" + (System.currentTimeMillis() % 1000000);
        AccountValue act0 = new AccountValue(userId, "password");
        // Change to accomodate the fact that CustomerValue does not include account information
        // CustomerValue cst0 = new CustomerValue( null, "firstnam", "lastnam", userId . . .
        // cst0.setAccountValue( act0 );
        CustomerValue cst0 = new CustomerValue(userId, "firstnam", "lastnam", userId + "@foo.com",
            "222-22", "f", "street2.", "street2.", "city", "0", "A1A-1A1", "C", "111-111-111",
            "Visa", "01-21");

        System.out.println("creating customer=" + cst0);
        // Update the method call for new signature
        String id = _petstore.createCustomer(cst0, act0);

        AccountValue act1 = new AccountValue(userId, "password1");
        CustomerValue cst1 = new CustomerValue(id, "firstname2", "lastname2", userId +
            "-2@foo.com", "222-222", "fr", "street2.1", "street2.2", "city2", "ON", "A2A-2A2",
            "CA", "222-222-222", "Amex", "02-22");
        // No such method on CustomerValues class any longer
        // cst1.setAccountValue( act1 );

        System.out.println("updating customer=" + cst1);
        // New signature for this method
        _petstore.updateCustomer(cst1, act1);

        CustomerValue cst = _petstore.getCustomer(id);
        // No such method on CustomerValues class any longer
        // AccountValue act = cst.getAccountValue( );
        AccountValue act = _petstore.getCustomerAccount(id);

        // !!!!!!!!!!!!!!!!!!! THE REST OF THE METHOD GOES HERE WHITHOUT CHANGE !!!!!!!!!!!!!!!!!!!
        assertEquals("firstname", "firstname2", cst.getFirstname());
        . . .
    }
}

```

18. After applying all these changes and rebuilding, you will see some warnings remaining; these are fixed by pressing **Ctrl+Shift+O** anywhere within the `PetstoreTest.java` file to automatically organize the imports.

19. Once you rebuild the project, you should not see any compilation errors.

The Web application is ready. As you can see, WRD recognized XDoclet Servlet tags in the `EJBTestServlet.java` file and generated the proper deployment descriptor for the Web application.

## Configuring resources for deployment into WebSphere

Before deploying the application into the WebSphere test environment, you need to create some of the resources, such as JDBC provider and datasource. In other chapters of the book we have used WebSphere administrative GUI to create those resources in the target runtime, but in this chapter we will use a new feature of WebSphere Application Server V6 called the extended EAR file.

What this means is that you can define JDBC resources within the xPetstore EAR file and when you deploy this EAR into the test or production runtime, those resources will be automatically created by WebSphere.

1. In the Project Explorer of the J2EE perspective of Rational Application Developer V6, open Enterprise Applications and expand the xPetstore-EAR project. Double-click **Deployment Descriptor: xPetstore-EAR**. This will open the deployment descriptor editor.

**Note:** When you expand the xPetstore-EAR project, you may see some errors. For this migration example, these errors are not relevant, so you may ignore them.

2. Go to the Deployment tab of the editor and under the JDBC provider list, click **Add**. Select values as described in the following table and click **Next**.

Table 7-3 JDBC provider values

Parameter	Value
Database type	IBM DB2
JDBC Provider Type	DB2 Legacy CLI-Based Type 2 JDBC Driver

3. In the Create a JDBC Provider window, define values as described in the following table and click **Finish**.

Table 7-4 Create JDBC Provider dialog

Parameter	Value
Name	DB2 JDBC provider

Parameter	Value
Implementation class name	COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
Classpath	<db2_home>\java\db2java.zip

**Tip:** For additional information about DB2 Universal Database V8.2 JDBC drivers, refer to the following URL:

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/ad/rjvjdapi.htm>

For additional information about the JDBC connectivity options supported by WebSphere Application Server V6, refer to the following URL:

[http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.wesphere.base.doc/info/aes/ae/rdat\\_minreq.html](http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.wesphere.base.doc/info/aes/ae/rdat_minreq.html)

- Now select DB2 JDBC provider from the JDBC Provider List and click **Add** from the *Data Source defined in the JDBC provider selected above* section. In the dialog, provide values as shown in the following table and click **Next**.

Table 7-5 Create a datasource dialog

Parameter	Value
Type of JDBC driver	DB2 Legacy CLI-based Type 2 JDBC driver
Datasource type	Version 5.0 datasource

- In the next window, provide the values as shown in the following table. JNDI name is the actual name in runtime. This will have to be mapped later to what our application uses via runtime bindings.

Table 7-6 Modify datasource dialog

Parameter	Value
Name	Petstore datasource
JNDI Name	jdbc/xPetstoreDatasource
Component managed authentication alias	DB2user
Use this datasource in CMP	Checked

Click **Next**.

6. In the next window, click **databaseName** and specify petstore in the Value: field. Click **Finish**.
7. Now you need to define the authentication alias for the JCA used in the datasource you just created. While still in the Deployment tab in the Deployment Descriptor editor, expand the **Authentication** section.
8. Click **Add** and provide values as shown in the following table and click **OK**.

Table 7-7 Create authentication alias

Parameter	Value
Alias	DB2user
User id	db2admin
Password	its0ra!0

9. Save the changes to the Deployment Descriptor. At this point, you have defined the runtime resource for our CMPs.
10. Now you need to add bindings between the application and the runtime resource. Open the Deployment Descriptor editor for the xPetstore-EJB project. In the Overview tab, under the JNDI - CMP Connection Factory Binding section, define the JNDI name to be the same as the runtime JNDI name for the datasource you have defined in the previous steps. In our example, we used jdbc/xPetstoreDatasource.
11. Save the changes to the EJB Deployment Descriptor.

## Configuring the test WebSphere server instance

For different projects within Rational Application Developer V6, it may be useful to define multiple WebSphere test servers independent from each other. Refer to Chapter 4, “Installation and configuration” on page 63 for details on how to create new WebSphere Profiles and servers. We assume that you already have defined a server in the Servers view on the J2EE perspective.

## Deploying and testing the application

We have completed the partial migration of the xPetstore EJB application and assembled it into a single EAR file. Before we proceed to deployment, make sure you rebuild the xPetstore-EAR project to add the latest updates into the EAR file.

Verify that there are no errors for projects xPetstore-EJB, xPetstore-EAR, and xPetstore-junit-WEB in the Problems view. If you enabled the filter then you have to click each of the projects to see errors or disable the filter to see a complete list of errors for all projects in your workspace.

If you see an error for the xPetstore-EAR project like the one shown in the following example, just ignore it, this message does not affect this migration example.

IWAE0037E The interfaces of the linked enterprise bean Petstore do not match those in EJB ref ejb/Petstore in module xPetstore-junit-WEB.war.application.xml  
xPetstore-EJB/EJB/META-INF

This version of xPetstore EJB has several automated tests built with JUnitEE. You are now ready to run these tests against the application. In the Project Explorer view expand **xPetstore-junit-WEB** → **WebContent** directory, right-click the **index.html** file, select **Run** → **Run on server** option. This will show a dialog window to let you select the test server instance.

Select the server you prefer and click **Finish**. Since you have defined your JDBC provider and datasource in the EAR file, you do not need to configure anything special in the test server. All resources will be created automatically at deployment time.

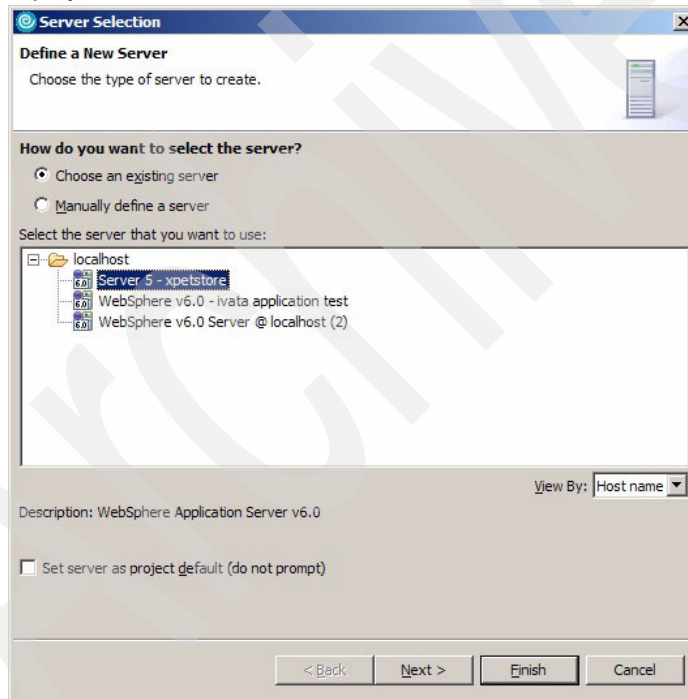


Figure 7-13 Selecting server for deployment and test



**Tip:** You can change your preferred server for the project if you right-click the project name, click **Properties** from the menu and click **Server** and then set the default server.

Once the server is started and the application is deployed, you are ready to test the application. A Web browser window will automatically open in Rational pointing to the JUnitEE TestRunner. Alternatively, you can open an external Web browser and access the following URL:

[http://localhost:9080/junit\\_test/](http://localhost:9080/junit_test/)

**Tip:** Having automated tests for the project would help in detecting problems that arise during the migration. Refer to Chapter 3, “Migration strategy and planning” on page 31 for information about how to implement a proper testing process for the application.

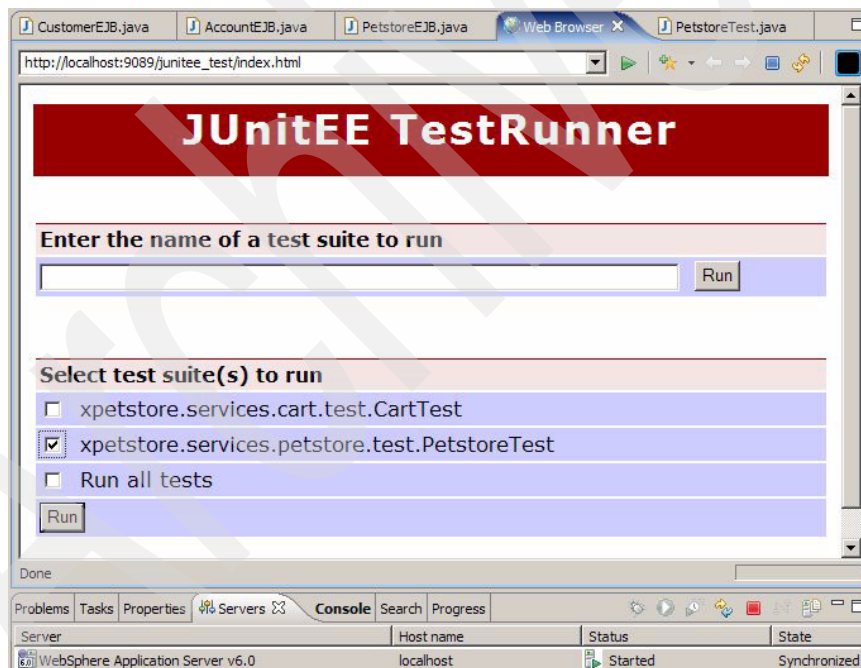


Figure 7-14 JUnitEE tests for the xPetstore EJB application

After you deploy the application into the running server, WebSphere generates runtime classes for Enterprise Beans. In our version of the product, this causes compilation errors in the files ConcreteCustomer\_948a290b.java and

ConcreteAccount\_0a6cb2fd.java right after we run the PetstoreTest; the error we receive is as follows:

```
Exception RemoveException is not compatible with throws clause in
AccountBeanImpl.ejbRemove()ConcreteAccount_0a6cb2fd.java
xPetstore-EJB/ejbModule/xpetstore/domain/signon/ejb line 48
```

If you open one of those files, you can see that WebSphere generates wrong method signature for the bean implementation:

```
public void ejbRemove() throws javax.ejb.RemoveException,
java.rmi.RemoteException {}
```

While according to the EJB 2.1 specification ([http://java.sun.com/j2ee/1.4/docs/api/javax/ejb/EntityBean.html#ejbRemove\(\)](http://java.sun.com/j2ee/1.4/docs/api/javax/ejb/EntityBean.html#ejbRemove())) the signature should be:

```
public void ejbRemove() throws RemoveException, EJBException {}
```

It is not clear why WRD has this problem since “normal” EJBs in Rational Application Developer V6 do not have this issue.

To fix this compilation error, as a temporary solution, we suggest that you add the following method to CustomerEJB.java and AccountEJB.java files:

```
public void ejbRemove() throws javax.ejb.RemoveException,
java.rmi.RemoteException {}
```

Once you add this method to both files, save and rebuild the project, all compilation errors should disappear.

Once more, check **xpetstore.services.petstore.test.PetstoreTest** and click **Run**. This should result in all tests executed successfully, as shown in Figure 7-15.

**Note:** We did not perform the migration of Cart EJB, so we will not run those tests.

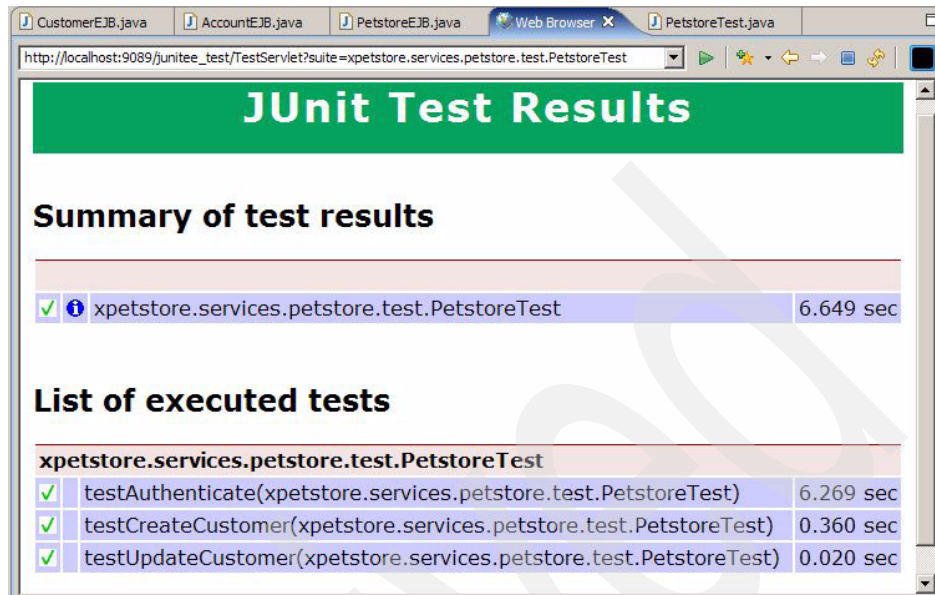


Figure 7-15 JUnitEE test cases completed for the migrated part of xPetstore EJB

**Note:** The first time you deploy the application into the server and run it, you may get a `NullPointerException` related to the JNDI lookup failure. If you receive such an error, click the **xPetstore-EJB** project and rebuild it. You may need to “touch” `CustomerEJB.java` in order for Rational to let you save and rebuild the project.

It is also suggested that you stop and restart the server to ensure a clean start and that the server be picking up the latest changes. This step is optional since Rational automatically redeploys the project to the test server and the test server will restart the newly rebuilt project.

Once you see the message in the console stating that the project was started, you can try running JUnitEE tests again. This time, you should not receive any errors.

## Summary

Unfortunately, WRD tags in WebSphere Application Server V6 are not fully compliant with XDoclet. Additionally, we have discovered several issues with code generation and had to work around those by changing Java code and generating different database schema as opposed to using the existing database

as is. This is the first release of annotation technology from IBM and it will continue to improve in future releases.

In this example, we were able to migrate part of the xPetstore EJB application from JBoss 3.2.7 into the WebSphere Application Server V6. As the migration tool, we have used Rational Application Developer V6 and the WRD facility of WebSphere. We have migrated the XDoclet tags to WRD tags with some changes for the following components:

- ▶ Two CMP EJBs
- ▶ One Session EJB
- ▶ Servlet

As you have seen during this migration not all of the XDoclet tags map directly into WRD tags and some Java code had to be changed because of the way WRD generated some of the artefacts in a different way from XDoclet. Despite the fact that WRD was modeled after XDoclet, it has some implementation differences that do not allow for transparent migration.

There are also some tags that are present in WRD that will not be found in XDoclet. It is to expect that, over time, WRD will be more closely aligned and compatible with XDoclet at the tag level. Another disadvantage of WRD is that it is not written as an extension to XDoclet and thus it may be hard to include WRD in a regular build process that is using Ant or Maven.

## 7.5 xPetstore EJB migration using Alternative 3

As we already discussed in 7.4, “xPetstore EJB migration using Alternative 2” on page 179, this application is a re-implementation of Sun’s PetStore blueprint application.

This application differs from the Sun version in that it uses open source frameworks extensively, including the XDoclet source code generation engine, which is used for generating both Java source code, vendor-specific deployment descriptors and J2EE standard deployment descriptors.

xPetstore EJB is an interesting application from a migration point of view because of the limitations in the XDoclet engine’s support for WebSphere Application Server V6 and J2EE 1.4.

**Note:** In section 7.4, “xPetstore EJB migration using Alternative 2” on page 179, we provide additional motivations for including xPetstore EJB in this book. We also document how it was migrated using a different approach and a different source code generation engine, WRD, which is the equivalent of XDoclet in Rational Application Developer V6.

### 7.5.1 Migration approach

As in all migrations projects, there are multiple approaches for migrating the application. By using the approach described here, we show how to migrate an application built using XDoclet and how to migrate an application without using Rational Application Developer V6. Applications that use XDoclet do not necessarily require all the features provided by Rational Application Developer V6. Instead, we use the Eclipse IDE, although any text editor can be used, and the Application Server Toolkit (ASTk) tool, which can be downloaded for free.

The Application Server Toolkit is a tool provided by IBM that can be used for assembling and deploying J2EE applications to the WebSphere Application Server V6 platform. The tool contains the same editors and functionality as Rational Application Developer V6 that are required for creating WebSphere Application Server V6 specific deployment descriptors.

**Note:** There are at least two options for obtaining and installing ASTk:

- ▶ The WebSphere Application Server V6 installation media includes a copy of ASTk that can be installed separately. Before starting the migration, we installed ASTk using the default settings provided by the installation wizard.
- ▶ WebSphere Application Server V6 Express also includes a copy of ASTk. A trial version of WebSphere Application Server V6 Express can be downloaded for free from:

<http://www.ibm.com/developerworks/downloads/ws/wasexp>

### 7.5.2 Configuring the initial environment

The steps required for setting up the xPetstore EJB initial environment, consisting of JBoss 3.2.7 and DB2 Universal Database V8.2, are documented in 7.4.2, “Configuring the initial environment” on page 181.

### 7.5.3 Migrating the xPetstore EJB application

In the following sections, we provide detailed, step-by-step instructions for how we performed the migration. We documented the issues we found and how we solved them.

#### Modifying the build script

xPetstore EJB uses a custom Ant build script for automating the build process. The Ant script executes different tasks including XDoclet and generation of the EAR package. The build script currently supports a couple of application servers but not WebSphere Application Server V6. Therefore, you have to modify the build scripts, which consist of the following files:

- ▶ `<source_home>\xpetstore-ejb\build.xml`  
This is the main build file which delegates some tasks to the application server specific build scripts.
- ▶ `<source_home>\xpetstore-ejb\build-<application_server>.xml`  
Named according to the target application server. Contains application server specific build scripts.
- ▶ `<source_home>\conf\as\appserver.properties`  
The configuration file is used for setting the target application server.
- ▶ `<source_home>\conf\as<application_server>.properties`  
Contains configuration parameters related to different application servers.
- ▶ `<source_home>\conf\db\database.properties`  
Database configuration; in this exercise, we do not change anything in these files. Instead, we use the ASTk to configure database-related information.
- ▶ `<source_home>\conf\db<database>.properties`  
Database-specific configuration. In our migration example, we do not modify this.

#### ***Adding support for WebSphere Application Server V6***

Add support for WebSphere Application Server V6 by modifying `build.xml`; this is done in the same manner as with the other supported application servers, mostly by copying application server specific tags and changing them to work with WebSphere Application Server V6.

It is also necessary to create a new file containing the WebSphere Application Server V6 specific XDoclet tasks, since XDoclet uses a set of different Ant tasks for each application server:

1. First, add the condition tag to the build.xml file as highlighted in the following example.

```
<source_home>\xpetstore-ejb\build.xml
```

*Example 7-21 Section of build.xml*

---

```
...
<condition property="weblogic">
  <equals arg1="${app.server}" arg2="weblogic" />
</condition>

<condition property="websphere">
  <equals arg1="${app.server}" arg2="websphere" />
</condition>

<!-- Paths -->
...
```

---

This will set the property websphere if the app.server property is equal to websphere. This is true in our case. This will tell the rest of the build script that our target platform is WebSphere

2. Add a call to the WebSphere Application Server V6 specific XDoclet target as highlighted in Example 7-22.

*Example 7-22 Section of build.xml*

---

```
...
  <antcall target="xdoclet.jboss" />
  <antcall target="xdoclet.orion" />
  <antcall target="xdoclet.weblogic" />
  <antcall target="xdoclet.websphere" />
</target>
...
```

---

This target will be defined in the following file that you will create later:

```
<source_home>\xpetstore-ejb\build-websphere.xml
```

The build script contains the following Ant tasks that are specific to WebSphere:

- websphere

This task provides support for IBM specific deployment descriptors used by EJBs and Servlets

For more information, refer to the XDoclet documentation available at the following URL:

<http://xdoclet.sourceforge.net/xdoclet/tags/ibm-tags.html>

- webspherewebxml

This task is used by XDoclet for generating the ibm-web-bnd.xmi and ibm-web-ext.xmi deployment descriptors for Web modules. Refer to the XDoclet documentation for more information:

<http://xdoclet.sourceforge.net/xdoclet/ant/xdoclet/modules/ibm/websphere/web/WebSphereWebXmlSubTask.html>

3. Create the WebSphere Application Server V6 specific configuration file as shown in Example 7-23 on page 232 in the following directory.

```
<source_home>\conf\as\websphere.properties
```

This file contains JNDI names that are used by the application.

*Example 7-23 WebSphere Application Server V6 build script configuration*

---

```
#=====
# WebSphere configuration file
#=====
# physical JNDI of the datasource
websphere.datasource=jdbc/xpetstoreDS
orion.datasource=jdbc/xpetstoreDS
# physical JNDI name of the javax.jms.MailSession
websphere.mail.session=mail/xpetstore/MailSession
orion.mail.session=mail/xpetstore/MailSession
# physical JNDI of the javax.jms.QueueConnectionFactory
websphere.queue.ConnectionFactory=jms/ConnectionFactory
orion.queue.ConnectionFactory=jms/ConnectionFactory
# physical JNDI of the queue used by xpetstore.services.order.OrderProcessorMDB
websphere.queue.order=jms/queue/mail
orion.queue.order=jms/queue/mail
# physical JNDI of the queue used by xpetstore.services.mail.MailerMDB
websphere.queue.mail=jms/queue/order
orion.queue.mail=jms/queue/order
# physical JNDI of the javax.jms.MailSession
websphere.mail.session=mail/MailSession
orion.mail.session=mail/MailSession
```

---

**Note:** The properties are defined twice in the configuration file because the xPetstore EJB source code is hardcoded to use properties named orion.

4. Add the Ant target file to the build.xml file as shown in Example 7-24. This target will call the XDoclet build script specific to WebSphere Application Server V6.



```
...
</target>

<target name="xdoclet.weblogic" depends="init" if="weblogic">
  <ant antfile="build-weblogic.xml" target="xdoclet" />
</target>

<target name="xdoclet.websphere" depends="init" if="websphere">
  <ant antfile="build-websphere.xml" target="xdoclet" />
</target>

<!-- ===== -->
...
```

---

5. Create the build-websphere.xml file containing the WebSphere-specific XDoclet Ant tasks as shown in Example 7-25 and place it in the following directory:

```
<source_home>\xpetstore-ejb\build-websphere.xml
```

This build script creates the IBM specific deployment descriptors for the EJB and Web application modules.

Example 7-25 Ant script that executes XDoclet - build-websphere.xml

---

```
<project name="xpetstore-servlet-orion" default="xdoclet" basedir="."/>

<!-- ===== -->
<!-- -->
<!-- Initialization -->
<!-- -->
<!-- ===== -->

<target name="init">

  <!-- General config -->
  <tstamp />
  <property environment="env" />
  <property file="${basedir}/build.properties" />

  <!-- Application server -->
  <property file="${conf.dir}/as/appserver.properties" />
  <property file="${conf.dir}/as/websphere.properties" />

  <!-- Paths -->
  <path id="xdoclet.class.path">
    <fileset dir="${lib.xdoclet.dir}" includes="**/*.jar" />
    <fileset dir="${lib.log4j.dir}" includes="**/*.jar" />
  </path>
</target>
```

```

        <fileset dir="${lib.common.dir}" includes="**/*.jar" />
        <fileset dir="${lib.j2ee.dir}" includes="**/*.jar" />
    </path>
</target>
<!-- ===== -->
<!-- -->
<!-- xDoclet -->
<!-- -->
<!-- ===== -->

<target name="xdoclet" depends="init">
    <antcall target="xdoclet.ejb" />
    <antcall target="xdoclet.web" />
</target>

<target name="xdoclet.ejb" depends="init">
    <taskdef
        name="ejbdoclet"
        classname="xdoclet.modules.ejb.EjbDocletTask"
        classpathref="xdoclet.class.path"
    />

    <ejbdoclet destdir="${build.java.dir}">
        <deploymentdescriptor destdir="${build.dir}/META-INF"
            useIds="true" validatexml="true"/>
        <fileset dir="${java.dir}" includes="**/*EJB.java" />
        <fileset dir="${java.dir}" includes="**/*MDB.java" />

        <websphere
            destdir="${build.dir}/META-INF"
            currentBackendId="DB2_BACKEND_ID"
            datasource="${websphere.datasource}"
        />
    </ejbdoclet>
</target>

<target name="xdoclet.web" depends="init" if="orion">
    <taskdef
        name="webdoclet"
        classname="xdoclet.modules.web.WebDocletTask"
        classpathref="xdoclet.class.path"
    />

    <webdoclet
        destdir="${build.dir}/WEB-INF"
        mergedir="${xdoclet.merge.dir}/web"
    >
        <deploymentdescriptor useIds="true"/>
        <fileset dir="${web.dir}" includes="**/*Filter.java" />
    </webdoclet>
</target>

```

```

        <fileset dir="${web.dir}" includes="**/*Servlet.java" />
        <webspherewebxml />
        </webdoclet>
    </target>
</project>

```

---

6. The WebSphere Application Server V6 specific deployment descriptors, which have the .xmi file extension, are not included in the EJB module by the build script. So you have to modify the build script.

Edit the build.xml file and replace the two *metainf* elements at the end of the Jar section with the code shown in the following example. This will tell Ant to include the .xmi files in the build.

*Example 7-26 Section of jar target in build.xml*

```

...
<!-- ===== -->
<!-- -->
<!-- Jar -->
<!-- -->
<!-- ===== -->
...
    <exclude name="**/web/**/*" />
    <exclude name="**/test/**/*" />
</fileset>

    <metainf dir="${build.dir}/META-INF">
        <include name="*.xml" />
        <include name="*.xmi" />
    </metainf>
    <metainf dir="${java.dir}/META-INF">
        <include name="*.xml" />
        <include name="*.xmi" />
        <exclude name="application.xml" />
    </metainf>
</jar>
...

```

---

7. Edit the appserver.properties file from the following directory:

```
<source_home>\conf\as\appserver.properties
```

and change the value of the app.server property to use WebSphere.

```
app.server=websphere
```

This tells the build script that the target platform is WebSphere.

**Important:** WebSphere Application Server V6 uses IDs to map the WebSphere-specific deployment descriptors to the standard ejb-jar.xml and web.xml deployment descriptors.

Therefore, you have to tell XDoclet 1.2.3 to generate IDs for each element in the deployment descriptors. This is done by setting to true the useIds attribute:

```
<deploymentdescriptor useIds="true"/>
```

8. Edit the <source\_home>\xpetstore-ejb\build.xml and configure all ejbdoclet and webdoclet elements to use IDs. This is done in the same way as build-websphere.xml, as highlighted in Example 7-25 on page 233.
9. In build.xml, disable the duplicates in the war task by adding a line as highlighted in the following example.

*Example 7-27 Disable duplicates in the war task*

```
...  
    <war  
        destFile="${dist.dir}/${war.name}"  
        webxml="${build.dir}/WEB-INF/web.xml"  
        duplicate="preserve"  
    >  
...  
...
```

### Updating XDoclet to version 1.2.3

XDoclet 1.2.3 contains new features and bug fixes for WebSphere Application Server V6.

1. Download the XDoclet 1.2.3 binary package from the following URL:  
<http://xdoclet.sourceforge.net>
2. Extract the contents of the package to a folder, we will use <xdoclet\_home> to refer to this folder.
3. Remove the content of the following directory:  
<source\_home>\lib\main\xdoclet
4. Copy all files from the following directory:  
<xdoclet\_home>\lib  
to:  
<source\_home>\lib\main\xdoclet
5. Make a new build by executing the following commands:

```
cd <source_home>\xpetstore-ejb
ant
```

This build will be imported later to the Application Server Toolkit (ASTk) to perform the CMP to database mapping. We will explain how to import the EAR file created by Ant into ASTk later.

## Configuring Mail Provider in WebSphere

The JavaMail session provides an abstraction layer for communicating with an e-mail server. This section describes how to configure a Mail Provider which is used to create a JavaMail session in WebSphere. Although it is not a necessary step, we created a new WebSphere profile to have a clean environment where we could deploy the newly migrated application. For information about how to create WebSphere profiles and port numbers, refer to Chapter 4, “Installation and configuration” on page 63.

1. Make sure the WebSphere Application Server is started and access the WebSphere Administrative Console at the following URL:  
`http://localhost:9060/ibm/console`
2. Select **Resources** → **Mail Providers** → **Built-in Mail Provider**. Click **Mail Sessions** in the Additional Properties section and create a new mail session with the following characteristics:

Table 7-8 JavaMail session configuration

Parameter	Value
Name	xPetstoreMail
JNDI name	mail/xpetstore/MailSession
Mail transport host	localhost
Mail transport protocol	SMTP
Mail store host	localhost
Mail store protocol	POP3

3. Click **OK** to create the mail session and save the configuration changes.

## Creating a new datasource in WebSphere

xPetstore uses a JDBC datasource for connecting to the database. To set up the datasource, you need to configure a JDBC provider, a J2C alias for storing the database authentication information and the datasource itself. The configuration is done from the WebSphere Administrative Console.

1. Select **Resources** → **JDBC Providers** and click **New**. Create a new JDBC provider with the following characteristics:

Table 7-9 JDBC provider configuration

Parameter	Value
Database type	DB2
Provider type	DB2 Legacy CLI-based Type 2 JDBC Driver
Implementation type	XA datasource

2. Click **Next**.
3. Specify the correct path to the DB2 JDBC driver db2java.zip and click **Apply**.
4. Click **Data sources** in the Additional Properties section and create a new datasource with the following characteristics:

Table 7-10 JDBC datasource configuration

Parameter	Value
Name	Petstore Datasource
JNDI name	jdbc/xpetstore
Database name	petstore

5. Click **Apply** and save your changes.
6. Click **Resources** → **JDBC providers** → **DB2 Legacy CLI-based Type 2 JDBC Driver (XA)** → **Data sources** → **Petstore Datasource** and click **J2EE Connector Architecture (J2C) authentication data entries** in the Related Items section then create a new alias with the following characteristics:

Table 7-11 J2C alias configuration

Parameter	Value
Alias	DB2user
User Id	db2admin
Password	its0ral0

7. Click **OK** and go back to the Petstore datasource. From the Component-managed authentication alias drop-down menu, select the J2C alias you just created.

8. Click **OK** and save your configuration.
9. Verify the connection to the database by selecting the datasource you just created from the list and click **Test connection**.

## Creating a new Integration Service Bus

The first step in configuring JMS messaging in WebSphere Application Server V6 is to create a new integration service Bus.

1. From the WebSphere Administrative Console select **Service Integration** → **Buses**.
2. Click **New** and specify bus as the name.
3. Click **OK** and save the changes to the configuration.

## Creating a bus member

The bus has to be deployed to a bus member which is an instance of WebSphere Application Server V6. Deploying the bus to a bus member will create a messaging engine in that WebSphere Application Server V6 instance. The bus member has its own data store for messages.

1. Select **Service Integration** → **Buses** and select the bus you just created.
2. Click **Bus members** in the Additional Properties section.
3. Click **Add** and select the server to where the bus will be deployed. Click **Next** and then **Finish**.
4. Save the changes to the configuration.

## Creating a bus destination

The destination is the target of all messages sent to the Integration Service Bus. To create the bus destination follow these steps:

1. Select the bus you created previously and click **Destinations** in the Additional Properties section.
2. Create a new queue destination. Specify xPetstore Queue as the identifier and deploy it to the bus member you created.
3. Click **Finish** and then save the configuration.

You have now configured the messaging bus, bus member and bus destinations, which together form the infrastructure for JMS messaging. The next step is to configure the JMS connection factory and queues.

## Creating a JMS connection factory

Before configuring the JMS queues, you need to configure a JMS connection factory which will be used by the xPetstore EJB MDBs to create a connection to the JMS provider.

1. Select **Resources** → **JMS Providers** → **Default messaging**.
2. Click **JMS queue connection factory** in the Connection Factories section.
3. Create a new JMS queue connection factory with the following characteristics:

Table 7-12 JMS queue connection factory configuration

Parameter	Values
Name	xPetstore JMS connection factory
JNDI name	jms/xpetstore/QueueConnectionFactory
Bus name	bus

4. Click **OK** and save the changes to the configuration.

## Creating JMS queues

xPetstore uses one Message Driven Bean for order processing and one for sending emails. The MDBs require that you set up two JMS queues.

1. Click **Resources** → **JMS Providers** → **Default messaging**.
2. Click **JMS queue** in the Destinations section.
3. Create a new JMS queue with the following characteristics:

Table 7-13 Mail queue configuration

Parameter	Value
Name	xPetstore Mail Queue
JNDI name	jms/queue/mail
Bus name	bus
Queue name	xPetstore Queue

4. Click **OK**.
5. Create a second JMS queue with the following characteristics:



Table 7-14 Order queue configuration

Parameter	Value
Name	xPetstore Order Queue
JNDI name	jms/queue/order
Bus name	bus
Queue name	xPetstore Queue

- Click **OK** and save the configuration.

### Creating the JMS Activation Specification for the JMS queues

The last thing you need to configure in WebSphere Application Server V6 related to JMS is the JMS Activation Specification that will bind the MDBs to the default JMS messaging provider.

The JNDI name of the JMS activation specification is used in the MDB's deployment descriptor to bind the MDB to the messaging queue.

- Select **Resources** → **JMS Providers** → **Default messaging**.
- Click **JMS activation specification** in the Activation Specification section.
- Create a new JMS activation specification with the following characteristics:

Table 7-15 Order MDB's JMS activation specification configuration

Parameter	Values
Name	xPetstore Order Activation Specification
JNDI name	eis/xPetstoreOrderActivation
Destination JNDI name	jms/queue/order
Bus name	bus

- Click **OK** and save the changes to the configuration.
- Create a second JMS activation specification in the same way with the following characteristics:

Table 7-16 Mailer MDB's JMS activation specification configuration

Parameter	Value
Name	xPetstore Mail Activation
JNDI name	eis/xPetstoreMailActivation

Parameter	Value
Destination JNDI name	jms/queue/mail
Bus	bus

6. Click **OK** and save the changes to the configuration.

You have now configured all the necessary components for JMS messaging.

## Creating deployment descriptors with ASTk

Application Server Toolkit (ASTk) is a tool provided by IBM that we show how to use, instead of Rational Application Developer V6, to create the WebSphere Application Server V6 specific deployment descriptors. We have to use ASTk because some deployment descriptors required by xPetstore EJB, that are specific to WebSphere Application Server V6, are not supported by XDoclet 1.2.3.

Although these deployment descriptors are written in XML, they are not easily edited by hand. This is the biggest difference compared to other applications servers, where hand edited deployment descriptors are common.

We show how to use ASTk for the following tasks:

- ▶ Map CMPs to an existing database schema  
This is the most complex task and it can, realistically, only be done with either Rational Application Developer V6 or ASTk. XDoclet 1.2.3 is not able to generate these files.
- ▶ Configure MDBs to use the JMS Activation Specification instead of listener ports

Earlier in this chapter, we described how to obtain a copy of ASTk.

1. Open the Application Server Toolkit *Navigator* view.
2. Click **File** → **Import** from the menu.
3. From the Import wizard window, click **Browse** and select the **xpetstore-ejb.ear** file that you built with Ant then click **Finish**.

This will import the EAR package and all its sub-modules into the workspace as separate projects.

You should now see the following projects in the Navigator view:

- ▶ xpetstore-ejb  
The xPetstore EJB EAR package. Contains all modules.

- ▶ xpetstore-ejb\_EJB  
EJB module
- ▶ xpetstore-ejb\_WEB  
Web application module
- ▶ xpetstore-ejb\_test  
Web application module containing unit tests

### ***Creating a new back end***

As described earlier, XDoclet 1.2.3 does not support the creation of WebSphere Application Server V6 back end configuration files. We considered two options for creating the back end.

#### ▶ Top-down

The top-down approach means that we would create a completely new database schema. This would require a data migration. If you choose the top-down approach, you have three options for creating the mapping:

##### – Rational Application Developer V6

Rational Application Developer V6 contains the Mapping Editor and the EJB to RDB mapping wizard, which can be used to generate a top-down mapping of the CMPs to a new database schema.

##### – Application Server Toolkit

The Application Server Toolkit contains the same mapping tools as Rational Application Developer V6. We chose to use this tool to demonstrate that there is no need to use the whole infrastructure provided by Rational Application Developer V6.

##### – Deployment time

WebSphere Application Server V6 will automatically create a top-down schema when you deploy your application, if you do not have the mapping configuration in your application package.

#### ▶ Meet-in-the-middle

You can reuse the existing database schema, by using the meet-in-the-middle approach. We chose this option because it does not require data migration. For this task we have two tools that we can use.

##### – Rational Application Developer V6

##### – Application Server Toolkit

Both contain exactly the same wizards and tools. As previously stated, we will use the Application Server Toolkit in this example.

**Note:** The database schema was originally created by JBoss 3.2.7 when xPetstore EJB was first deployed.

Next, we describe how to generate the database mapping for the CMPs. We use ASTk, which contains the same database mapping tools as Rational Application Developer V6, and the meet-in-the-middle approach since we already have a database that we want to with the migrated application:

1. Open Application Server Toolkit Navigator view and right-click the **xpetstore-ejb\_EJB** project and select **EJB to RDB Mapping** → **Generate Map**.
2. Select the **Create a new backend** folder and click **Next**.
3. Select **Meet-in-the-middle** and click **Next**.
4. Create a database connection with the following characteristics:

Table 7-17 Database configuration for wizard

Parameter	Value
Connection name	xpetstore connection
Database	petstore
User ID	db2admin
Password	its0ral0
Database vendor type	DB2 Universal Database Express V8.2
JDBC Driver	IBM DB2 APP DRIVER

5. Click **Next**, select all tables in the DB2ADMIN schema from the list.
6. Click **Next**, select **Match By Name, and Type**.  
This will try to match CMP fields to database tables by looking for similarities in name and data type.
7. Click **Finish**.

Since the table names and the CMP field names do not match, the mapping wizard is not able to map any of the tables. The tree on the left contains the CMPs and their fields, while the tree on the right shows the database tables.

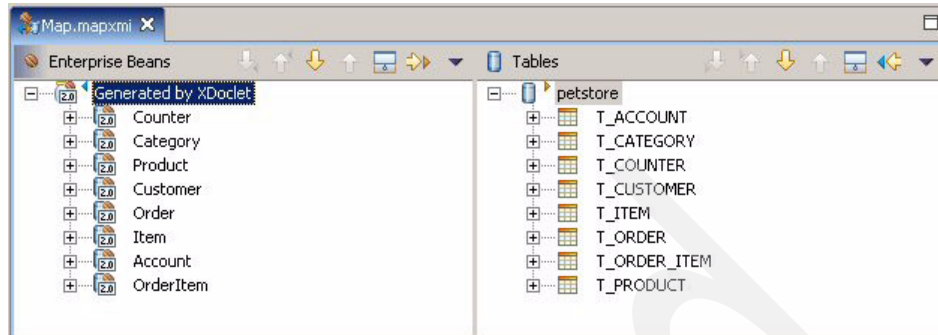


Figure 7-16 Incomplete mapping shown in the Mapping Editor

### ***CMPs to tables manual mapping***

To manually map the CMPs to the database, first map the Counter bean to the T\_COUNTER table.

1. Select the **Counter** bean from the tree to the left and the **T\_COUNTER** table from the tree to the right.
2. Right-click the **T\_COUNTER** table and select **Create Mapping**.  
You should notice that the icons have changed to indicate that the mapping was successful.
3. Map each of the remaining unmapped CMPs to the correct tables in the same way as described in the previous step.
4. Map each of the CMP fields to a table column the same way as you mapped the the CMPs and tables. Select the CMP field and the corresponding table column to be mapped to. Right-click the table and select **Create Mapping** as illustrated in Figure 7-17.

**Note:** You can tell the Mapping Editor to automatically map the fields. Select the CMP bean and the table that you want to map automatically. Right-click the CMP and select **Match by Type**.

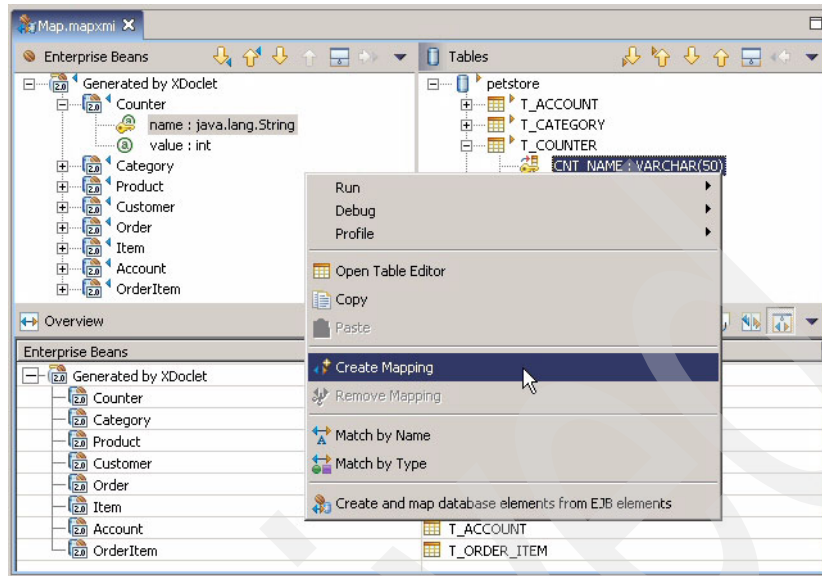


Figure 7-17 Mapping a CMP field to a table column

5. You also have to map the relationships. This is done either manually or with the *Create and map database elements to EJB elements* wizard.

We used the *Create and map database elements to EJB elements* wizard to have all the relationships automatically mapped.

To use the wizard, select the top hierarchy entry, **Generated by XDoclet**, on the Enterprise Beans section (left side) and right-click the database, **petstore**, from the Tables section. From the pop-up menu, select **Create and map database elements to EJB elements**.

The tool will create a couple of new columns, as illustrated in Figure 7-18.

The tool uses these new columns to map some of the relationships. This is because the column names are not the same on both sides of the relationship.

We will map the relationships to the correct columns and then remove the columns that the wizard created.

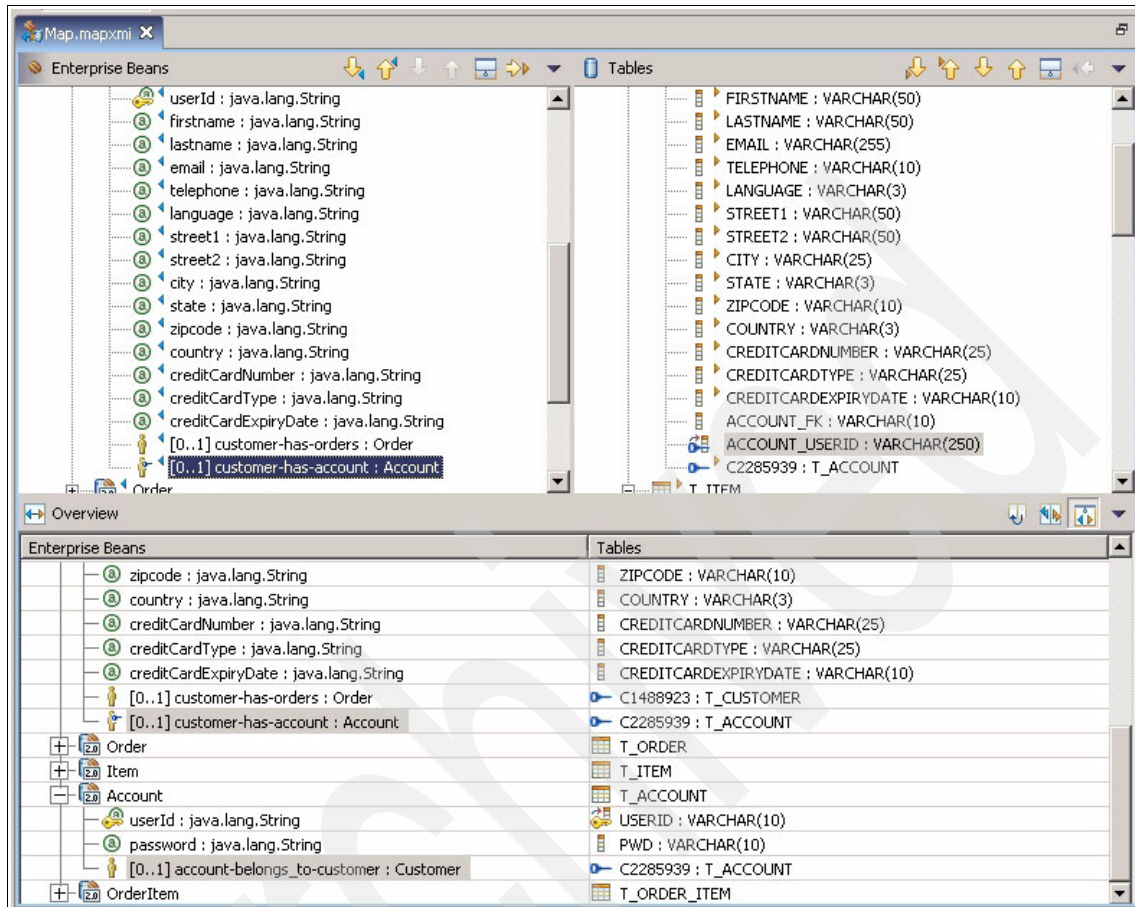


Figure 7-18 CMR relationship customer-has-account for the Customer CMP shown in the Mapping Editor

6. Right-click the **T\_CUSTOMER** table and select **Open Table Editor**.
7. Select the foreign key that was created by the wizard.
8. Select the **ACCOUNT\_USERID** column and click the appropriate button to remove it.
9. Map the **ACCOUNT\_FK** column to the foreign key, by adding **ACCOUNT\_FK** to the list. This column is the original column used in the relationship, but it was not mapped because the name is not the same on both sides of the relationship.
10. Next, go to the Columns tab and delete the **ACCOUNT\_USERID** column.
11. Save the changes and close the Table Editor.

12. Repeat the steps for the T\_ITEM, T\_ORDER, T\_ORDER\_ITEM, T\_PRODUCT tables. Remove the columns created by the wizard and map the relationships as explained in the previous step.
13. Verify that the back end we just created is active by double-clicking the ejb-jar.xml file in the Navigator view. This opens the Deployment Descriptor Editor. On the Overview tab scroll down to the Backend ID section and verify that it is using DB2 Universal Database V8.2 as illustrated in Figure 7-19.

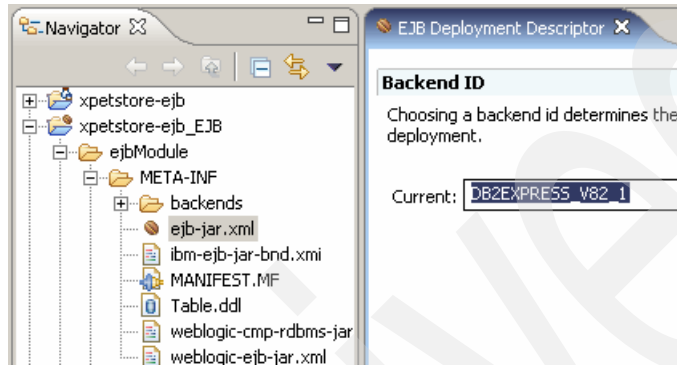


Figure 7-19 EJB Deployment Descriptor

You have now created and configured the back end. Next, we will describe how to modify the build so that it includes the new back end.



**Tip:** The Meet-in-the-middle Mapping wizard is not able to map CMPs to tables correctly if the table and column names do not match the CMP field names. You can still use the wizard by helping it and executing it in increments:

1. Map tables to CMPs automatically with Meet-in-the-middle wizard  
Tables that the wizard cannot map will be shown when you activate **Show both the mapped and unmapped objects**.
2. Map any unmapped tables manually  
When you have all tables mapped correctly, you can use the **Match by Name** or **Match by Type** mapping tools to map the columns.
3. Map columns either with **Match by Name** or **Match by Type**.  
This will try to match the CMP fields to table columns using the name and data type.
4. Map any unmapped columns manually  
Select the column and right-click the CMP field, then select **Create Mapping** from the menu.
5. Map relationships automatically by using **Create and map database elements to EJB elements**.  
The tool will map all relationships it can automatically. For the relationships that it cannot map, because of name or data type differences, it will create new columns. We can use the example that the tool creates to map the relationships. After the mapping is done, we can remove the new columns that were created by the tool.
6. Remap foreign keys to correct columns  
The columns that the tool created can be removed. But first, we have to map the relationships to the correct column. Open the Foreign Keys page in the Table Editor and map the foreign key to the correct column.
7. Remove columns that the tool created  
Next, we delete the new column from the database schema by going to the *Columns* page and removing the new column.

## Configuring MDBs

XDoclet 1.2.3 does not support J2EE 1.4 and WebSphere Application Server V6 specific features. We can only use XDoclet 1.2.3 for mapping MDBs to a listener port. Since this is not possible with the default messaging engine in WebSphere

Application Server V6 we have to map the MDBs to a JMS activation specification manually using the ASTk.

1. Open the Application Server Toolkit Navigator view.
2. Open ejb-jar.xml with the EJB Deployment Descriptor Editor. Go to the Bean tab.
3. Select the **OrderProcessor** MDB from the list of EJBs.
4. In the WebSphere Bindings section, select **JCA Adapter** and specify the following values:

Table 7-18 JCA Adapter configuration for the Mailer MDB

Parameter	Value
ActivationSpec JNDI name	eis/xPetstoreOrderActivation
Destination JNDI name	jms/queue/order

5. Select the **Mailer** MDB from the list of EJBs.
6. In the WebSphere Bindings section, select **JCA Adapter** and specify the following values:

Table 7-19 JCA Adapter configuration for the OrderProcessor MDB

Parameter	Value
ActivationSpec JNDI name	eis/xPetstoreMailActivation
Destination JNDI name	jms/queue/mail

**Tip:** You can also configure the MDBs to use the JMS activation specification, by using the WebSphere Administrative Console at deployment time.

The configuration is done on the following page: **Enterprise Applications → xPetstore - EJB → Provide JMS and EJB endpoint URL information → Binding enterprise beans to listener port names or activation specification JNDI names.**

## Exporting the EAR application from ASTk

The last step before deploying the application is to create an EAR file with Application Server Toolkit. We do this by using the Export wizard in the ASTk. This wizard is the same as the one in Rational Application Developer V6.

1. Still in the Application Server Toolkit Navigator view, right-click the **xpetstore-ejb** EAR project and select **Export**.

2. From the Export pop-up menu, select **EAR file** and click **Next**.
3. On the next page, specify the following as the file name:  
`<source_home>\xpetstore-ejb.astk.ear`
4. Click **Finish**.

## Deploying the application

Make sure that WebSphere Application Server V6 is started.

1. Open the WebSphere Administrative Console by accessing the following URL:  
`http://localhost:9060/ibm/console/`
2. Select **Applications** → **Install New Application**.
3. Specify the path to the xPetstore EJB EAR file:  
`<source_home>\xpetstore-ejb.ear`
4. Click **Next** and select **Generate Default Bindings**.
5. Continue to the next page and click **Step 13: Summary**, leaving all the previous steps with the default values.
6. Click **Next** and then **Finish**. The application should deploy with some warnings but without errors.
7. Save the changes configuration.
8. Select **Applications** → **Enterprise Applications**.
9. Select the check box for the **xPetstore - EJB** application and click **Start**.

You have now successfully built, deployed and started the xPetstore EJB application, the next step is testing.

## Testing the application

The xPetstore EJB application includes one Web application that contains JUnitEE unit tests. This makes it easier for us to test the application and verify that it is working after the migration has finished. To run the unit tests, open the following URL in a browser:

`http://localhost:9080/xpetstore-test/index.html`

On the page, you will see a list of unit tests. Select each unit test and click **Run**. If all tests were successful, you should see the test result page with no error, as shown in Figure 7-20 on page 252.

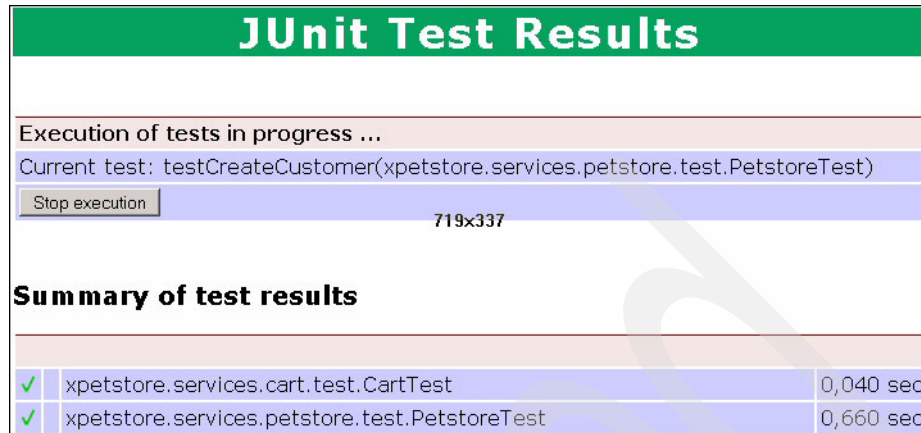


Figure 7-20 JUnitEE test cases completed for the migrated part of xPetstore EJB

## Modifying the build script to include the back end

Using ASTk each time for creating the deployment descriptors and CMP mappings is time consuming. Instead, it is more practical to generate the deployment descriptors once with ASTk and then modify the Ant build script to include them.

In this section, we describe how to do this by including the new back end in the build created by Ant and configuring the deployment descriptors generated by Ant to use this back end.

**Note:** You will have to map the MDBs to the JMS Activation Specification at deployment time using the WebSphere Administrative Console, since the WebSphere Application Server V6 descriptors used by the MDBs are regenerated each time you run the Ant build script. The back end configuration is located in separate files and are not touched by Ant.

1. Open the Navigator view in ASTk and expand the **xpetstore-ejb\_EJB** project.
2. Copy the content of the following directory:  
`xpetstore-ejb_EJB\ejbModule\META-INF\backends`  
to:  
`<source_home>\xpetstore-ejb\conf\META-INF`  
You will have to create the \conf\META-INF directory.

Modify the build script to include the back end in the EJB jar by adding the following lines to the build.xml file in the EAR section as highlighted in Example 7-28 .

The added jar task updates the EJB jar with the contents in the following folder:

```
<source_home>\xpetstore-ejb\conf\META-INF
```

*Example 7-28 Excerpt from build.xml*

---

```
...
<!-- ===== -->
<!-- -->
<!-- Ear -->
<!-- -->
<!-- ===== -->
<target name="ear" depends="init">
    <echo>+ ===== +</echo>
    <echo>+ +</echo>
    <echo>+ Building ear files +</echo>
    <echo>+ +</echo>
    <echo>+ ===== +</echo>

    <delete>
        <fileset dir="${dist.dir}" includes="${ear.name}" />
    </delete>

    <jar update="true" duplicate="preserve"
        destfile="../dist/xpetstore-ejb.jar" basedir="conf">
        <fileset dir="conf" includes="**/*" />
    </jar>

    <ear
        destfile="${dist.dir}/${ear.name}"
        appxml="${java.dir}/META-INF/application.xml"
    >

...

```

---

3. You also have to tell XDoclet to use the back end we created with ASTk. This is done by specifying the back end ID in the *websphere* Ant task's *currentBackendId* attribute in the following file:

```
<source_home>\xpetstore-ejb\build-websphere.xml
```

The contents of this file are displayed in Example 7-29.

```
...
<websphere
  destdir="${build.dir}/META-INF"
  currentBackendId="DB2EXPRESS_V82_1"
  datasource="${websphere.datasource}"
/>
...
```

---

**Note:** You can find the back end's ID by opening the EJB Deployment Descriptor editor and going to the Overview page. In the WebSphere Bindings section there is a Backend ID heading where the back end ID can be found.

## Summary

In this migration example, we were able to migrate xPetstore EJB successfully, maintaining both the original development environment and database schema. In the migration, we demonstrated how to use:

### ► XDoclet 1.2.3

XDoclet generates most of the IBM specific deployment descriptors that are needed. The only two exceptions in our example are:

#### – JMS Activation Specification

We cannot use the listener port with the WebSphere Application Server V6 default messaging engine since XDoclet 1.2.3 does not support the JMS Activation Specification. Hopefully, this feature will be added to future versions. There is a bug report related to this which can be found at the following URL:

<http://opensource.atlassian.com/projects/xdoclet/browse/XDT-1401>

#### – CMP mapping

XDoclet 1.2.3 is not able to create the database mapping for CMPs. This task is complex so we don't expect this problem to be solved easily.

### ► Application Server Toolkit V6

We used the Application Server Toolkit V6 for the tasks that XDoclet 1.2.3 did not support.

#### – Meet-in-the-middle

We used the mapping EJB to RDB mapping wizard to map CMPs to database tables, which was by far the most difficult migration issue.

- EJB Deployment Descriptor editor

We used the EJB Deployment Descriptor editor to specify the JMS Activation Specification for the MDBs.

Archived

Archived



## Migrating from Tomcat

In this chapter, we describe how to migrate two J2EE 1.3 Web applications from Apache Tomcat 5.5.9 to WebSphere Application Server V6. There are several challenges associated with such migrations. One of them is the fact that Tomcat applications are often built on top of different open source frameworks and libraries, such as Apache Commons, Hibernate, Spring and many others.

Naturally, there is a concern that those frameworks may not work on commercial applications servers. The other challenge is that the development and build environment is often command line based and uses Ant or Maven tools with no dependency on any particular IDE. At the same time, the perception among developers is that IBM development tools are mandatory to build WebSphere applications. We will demonstrate later on in this chapter that this is not the case.

When we started this book, we ran a search on the Web for open source Java Web applications. We have selected two typical Tomcat applications that we found on sourceforge.net to reflect real world experience during migration.

We also took Servlet 2.4 and JSP 2.0 samples included in the Apache Tomcat 5.5.9 distribution and simply redeployed them into WebSphere Application Server V6 without changing a single line of Java or XML code. This took only a few minutes and should not be classified as a “migration.” That is why we did not include those in this book.

## 8.1 Introduction

Apache Tomcat 5.5.9 is an open source implementation of Sun's J2EE Web container. It implements Java Servlet and JSP specifications and is used in the official Reference Implementation for the Java Servlet and JavaServer Pages specifications. Tomcat is developed by a community of developers under the umbrella of the Apache Software Foundation (ASF). Tomcat is widely used by large number of Web sites. It is popular for its small footprint, simplicity and large number of toolkits and frameworks that run on it. One of its nice features is the fact that it is open source and it is free to use.

Applications tend to grow over time and at some point need higher quality of services (QoS) and more sophistication than Tomcat can offer, such as centralized administration for clustered domains, failover, advanced role-based security for users and administrators, script-based administration, advanced monitoring tools, portal infrastructure, page fragment and distributed Java caching, transactions, etc. That is why some applications need to be redeployed or migrated into enterprise grade application servers, such as WebSphere Application Server V6.

The two applications we have selected for migration are built on top of a number open source frameworks, as illustrated in Table 1-2 on page 4; these applications are:

- ▶ ivata groupware version 11.1
- ▶ xPetstore Servlet version 3.1.3

## 8.2 Prerequisites and assumptions

There are some prerequisites and assumptions we need to consider before carrying on with the migration examples. One basic requirement is that the reader should have the "destination" environment installed and configured as described in Chapter 4, "Installation and configuration" on page 63. This should provide some initial background knowledge if the reader is not yet familiar with IBM products.

The reader should also be familiar with the J2EE specification and architecture, be able to understand and run basic SQL commands. The reader should also be familiar with Rational Application Developer V6, Ant and Maven. Last but not least, the reader should be knowledgeable about Apache Tomcat 5.5.9.

The following software should be installed before starting the migration:

► J2SE 1.4

We use J2SE 1.4 since it is supported by WebSphere Application Server V6 and we need to make sure our applications work well in this JVM. The WebSphere Application Server V6 installation includes JDK 1.4.2 and you do not need to install additional JDKs on your machine.

► Apache Tomcat 5.5.9

This version was chosen because it was the latest version available at the time of writing this redbook. The Tomcat home page advises all Tomcat users to upgrade to Tomcat 5.x whenever possible.

► Ant 1.5.1

Apache Ant is a Java-based build tool. It is similar to the Make tool, but has many enhanced features that make it very good choice for Java applications. This is required for migrating the xPetstore Servlet application.

► Apache Maven 1.0.2

Maven is an open source tool that extends Ant and provides software project management capabilities. This is only required for migrating an ivata groupware application.

► hMailServer 3.4.1

This is needed for ivata groupware on Windows platform only. This server comes with MySQL database embedded and will install it as Windows service.

► IBM WebSphere Application Server V6

Before you start migrating to WebSphere Application Server V6, we recommend that you play with sample applications shipped with the product and also read publications listed in “Related publications” on page 325. Instructions on how to install and configure WebSphere Application Server V6 are covered in Chapter 4, “Installation and configuration” on page 63.

► IBM UDB DB2 8.2

WebSphere Application Server V6 supports many databases, but we have decided to use DB2 for all our applications. Instructions on how to install and configure DB2 Universal Database V8.2 can be found in Chapter 4, “Installation and configuration” on page 63.

► Rational Application Developer V6

Rational Application Developer V6 was only used for the xPetstore migration project covered in 8.5, “xPetstore Servlet migration” on page 280. Instructions on how to install and configure Rational Application Developer V6 can be found in Chapter 4, “Installation and configuration” on page 63.

## 8.3 Software installation

This section provides high-level steps and tasks for installing and configuring Tomcat, Maven, Ant and some other products to be able to run our sample applications in their original environment. Instructions on how to install and configure the WebSphere destination environment are covered in detail in Chapter 4, “Installation and configuration” on page 63.

### 8.3.1 Apache Tomcat

Detailed instructions for installing, configuring and managing Apache Tomcat 5.5.9 are provided in the product documentation guides available at the following URL:

<http://jakarta.apache.org/tomcat/tomcat-5.5-doc/index.html>

The following list highlights the general tasks we performed to install and configure our initial environment as a starting point for deploying sample applications.

1. Download Apache Tomcat 5.5.9 binary distribution for Windows from the following URL:

<http://archive.apache.org/dist/jakarta/tomcat-5/v5.5.9/bin>

Here is the list of modules that we have downloaded (total of 9.6 MB):

- jakarta-tomcat-5.5.9.zip (base Tomcat install)
- jakarta-tomcat-5.5.9-compat.zip (need this to run Tomcat on JDK 1.4)
- jakarta-tomcat-5.5.9-admin.zip (admin application)

2. Unzip all archives into the same directory (later referred to as <tomcat\_home>):

```
c:\tomcat
```

3. Edit <tomcat\_home>\bin\setclasspath.bat file to update it for the JDK that you are planning to use with your Tomcat install. Since we have installed a compatibility package, we prefer to run Tomcat on the same JDK as WebSphere. This will help us to ensure that our sample application runs on the same JDK in both environments and may eliminate potential JDK compatibility issues (unlikely, but possible, especially considering that Apache Tomcat 5.5.9 runs on J2SE 1.5). Add the following lines at the beginning of the file:

```
set JAVA_HOME=<was_home>\java
set BASEDIR=C:\tomcat
```

4. Install the DB2 JDBC driver. The DB2 driver will be used by our applications for connecting to DB2 Universal Database V8.2. Copy db2java.zip from:

<db2\_home>java

to the following directory:

<tomcat\_home>\common\lib

5. In the <tomcat\_home>\common\lib\ directory, rename db2java.zip to db2java.jar.

**Note:** You have to rename the file containing the database driver from .zip to .jar. Tomcat does not put .zip files in the classpath automatically.

6. You are now ready to start the Apache Tomcat 5.5.9 server. Go to your <tomcat\_home>\bin directory and run the following command:  
startup.bat
7. Once the server is started, you can verify that it is running by opening a Web browser and pointing it to this URL: <http://localhost:8080>. You should see the Tomcat Welcome window.

### 8.3.2 Apache Maven

Although Apache Maven covers more functionality than just building, we are using it only with that goal in mind. We only used Apache Maven for migrating the ivata groupware application. We used Apache Ant for xPetstore Servlet.

Instructions on how to download, install and configure Maven 1.0.2 can be found at the following URL:

<http://maven.apache.org>

The Apache Maven installer does not define the Maven directory in your path. Make sure you define the Maven binaries directory in your path to be able to run Maven from your source directory.

### 8.3.3 Apache Ant

Apache Ant is a Java-based build tool widely used in Java and J2EE projects. Most of the applications we worked with, when writing this book, used Apache Ant to create their ear or war files. Instructions on how to download, install and configure Ant 1.5.1 can be found at the following URL:

<http://ant.apache.org>

Once again, make sure you define the Ant binaries directory in your path to be able to run Ant from your source directory. We will refer to the **ant** command as if it were defined in the path.

### 8.3.4 hMailServer

The ivata groupware application implements some mail functionality, so it will need a mail server installed. The ivata groupware suggested mail server is hMailServer. Instructions on how to download, install and configure hMailServer can be found at the following URL:

<http://www.hmailserver.com/>

It should take you only few minutes to set up and get hMailServer running. Additional configuration steps are covered in 8.4.2, “Configuring the source environment” on page 263.

You can find additional information about hMailServer and ivata groupware can be found at the ivata groupware URL:

<http://groupware.ivata.org>

## 8.4 ivata groupware migration

The ivata groupware application implements team collaboration functionality such as calendar, e-mail client, address book, FAQ, discussions, library of documents, etc. It works on Unix, Linux and Windows, is implemented as a J2EE 1.3 Web application and supports JBoss 3.2.x and Tomcat 5.x (the same war file can be run on either server). This application is distributed with the full source code under the terms of the GNU General Public license.

We found this to be an interesting application to be migrated for the following reasons:

- ▶ It is based on popular open source frameworks (refer to Table 1-2 on page 4 for further details on frameworks).
- ▶ It was designed for running on Tomcat and JBoss.
- ▶ It was never tested on WebSphere Application Server V6.
- ▶ The build process uses Maven, a very popular build and software project management tool. This is typical for applications that are built for Tomcat and JBoss.
- ▶ The application is built using plain old Java objects (POJO) instead of EJBs for the business objects and its source code is not tied to any specific IDE.
- ▶ The application is far from trivial, has advanced functionality and is fairly large in size. It uses J2EE and J2SE functionality such as JSP 1.2 with a number of custom tags, Servlet 2.3, JSTL 1.0.2, JavaMail 1.3.2, etc.
- ▶ The application uses the COM interface to access the Windows application, hMailServer.

- ▶ ivata groupware is actively developed. At the time of writing this book, we were using the last available update for this application, released in April 2005.

Overall, we believe this application is representative of what a typical ISV would build if they were using Tomcat or JBoss as their deployment platform.

### 8.4.1 Migration approach

The ivata groupware application was built with non-IBM development tools using Maven as a build tool and various editors for creating Java and XML files, partly generated by XDoclet and other frameworks. The application will have to be enhanced in the future and it needs to support Tomcat as well as WebSphere as deployment targets. The decision has to be made about what tool to use to migrate the application and what tools will be used to continue development of this application going forward. Different alternatives for migrating this application are shown in Figure 3-1 on page 52.

For this particular migration project, we have decided to go with 3.4.1, “Alternative 1: Keeping the existing development environment” on page 52. Since the typical application development environment for the Tomcat “shop” is represented in Alternative 1, we have decided to use this approach for migrating and maintaining the application. The existing application was built by a team who is already familiar with the current build and development environment and has best practices built around it. It may be more productive to take advantage of their current skills and continue using existing processes, frameworks and tools, assuming they work well for both environments, Tomcat and WebSphere.

Instead of Rational Application Developer V6 or any other IDE, we used Eclipse with a few plugins (such as XMLBuddy and others) as a text and XML editor. We wanted to show that it is possible to migrate an application, like ivata groupware, to WebSphere Application Server V6 without using an IDE. We also wanted to demonstrate that it is possible to maintain the original development environment.

### 8.4.2 Configuring the source environment

It is important to make sure that the application works properly in the source environment before we try to migrate it to WebSphere. This section describes how we configured Apache Tomcat 5.5.9 and ivata groupware to run in the source environment. Instructions on how we installed the Apache Tomcat 5.5.9 environment are described in 8.3, “Software installation” on page 260.

## Getting the ivata groupware application

Before starting with the migration, you need to obtain source files for the ivata groupware application. You can download the file `ivatagroupware-src-0.11.1.zip` with all source files from the following URL:

<http://groupware.ivata.org>

Create a source home directory and extract the application source code you just downloaded. From now on, we will refer to this directory as `<source_home>`.

## Installing and configuring prerequisites

For the most updated information about installation and configuration prerequisites for the ivata groupware application, including `hMailServer` and `Maven`, refer to the application documentation at the following URL:

<http://groupware.ivata.org/install.html>

## Building the ivata groupware application

Detailed information about how to build the application war file from sources can be found directly in the application Web site at the following URL:

<http://groupware.ivata.org/develop/build.html>

In a nutshell, once `JAVA_HOME` is defined, the ivata sources are unzipped and `Maven` is installed, you need to run the following command from the `<source_home>` directory:

```
maven
```

Assuming that the build is completed without errors, the resulting war file will be located in the following directory:

```
<source_home>/package/war/target/ivatagroupware-war-0.11.1.war
```



**Note:** We experienced some problems while building the application. Those problems were related to a missing jar file in Maven. We were receiving the following warnings while building the ivata groupware application:

```
Tag library requested that is not present: 'maven' in plugin:
'maven-xdoclet-plugin-1.2'
Attempting to download xjavadoc-1.0.2.jar.
WARNING: Failed to download xjavadoc-1.0.2.jar.

BUILD FAILED
File..... C:\Documents and
Settings\<user>\.maven\cache\maven-multiproject-plugin-1.3.1\plugin.jelly
Element... maven:reactor
Line..... 217
Column.... 9
Unable to obtain goal [multiproject:install-callback] --
D:\sampleApp\ivatagroupware-src-0.11\hibernate\maven.xml:92:49: <attainGoal>
The build cannot continue because of the following unsatisfied dependency:

xjavadoc-1.0.2.jar

Total time: 45 seconds
```

We solved this problem by copying the xjavadoc-1.0.2.jar file from the following directory:

```
C:\Documents and Settings\<user>\.maven\repository\xdoclet\jars\

to:

C:\Documents and Settings\<user>\.maven\repository\xjavadoc\jars\
```

## Deploying the application

Once you have installed all prerequisites and built the application war file, you can deploy application into Tomcat. Tomcat will automatically deploy Web applications located in the webapps directory by default. Follow these steps to deploy the ivata groupware application:

1. Copy the WAR file from the following directory:

```
<source_home>\package\war\target\ivatagroupware-war-0.11.1.war

to:

<tomcat_home>\webapps
```

2. From the command line, change to the Tomcat directory:

```
<tomcat_home>\bin
```

3. If you have already deployed ivata groupware into Tomcat (let's say you made several attempts), you need to stop Tomcat by running the following command:

`shutdown.bat`

You also need to remove the previously deployed application from the `<tomcat_home>\webapps` directory. Make sure to remove both the `ivatagroupware-war-0.11.1.war` file and the `ivatagroupware-war-0.11.1` directory (exploded war file).

4. Start Tomcat by running the following command:

`startup.bat`

You should see no errors in the log if the application was deployed successfully.

5. Make sure that you have JavaScript enabled in your Web browser.
6. Tomcat builds a root URI for the Web application based on the name of the war file. Therefore, you can access the application at the following URL (assuming you kept default port for the Tomcat installation and did not rename the war file):  
  
`http://localhost:8080/ivatagroupware-war-0.11.1`  
  
To log in to the application, you can use username `ivata` and password `changeit`.
7. A window appears showing a message about the mail server configuration. You need to configure ivata. Click **setup program** to configure your mail server and database properties.
8. Select **HSQL** as your database, accept all defaults and click **OK**. ivata groupware will populate the HSQL database and then show you the main window of the application.

**Note:** The ivata groupware application is very sensitive to the mail server configuration. If you have any problems with the application, check the mail server configuration first and make sure the mail server works properly.

## Testing the application

The application we just deployed uses an HSQL database that is included with the source download of the ivata groupware application.

This version of ivata groupware has several automated tests. Additional tests can be implemented, for example with JUnit, HttpUnit or Cactus.

**Tip:** Having automated tests for the project would help in detecting problems that may arise during the migration. Refer to Chapter 3, “Migration strategy and planning” on page 31 for information about how to implement a proper testing process for the application.

Additionally, we tested the application manually, to verify all the functionality of the application. We logged in the application, selected the calendar option and added some events on different days. We browsed all the events using the arrows in the calendar, and selected one to modify its content. Then we browsed the contact list and added a contact.

After that simple test, we checked the log files and did not find any errors. This simple test was implemented just to ensure that the application was up and running without configuration problems. This is obviously not recommended for a production environment.

At this point, we have successfully built, deployed and tested ivata groupware running on Apache Tomcat 5.5.9, which is its original environment. Having the application working properly on the source environment is our starting point for migrating it to WebSphere Application Server V6.

### 8.4.3 Migrating the ivata groupware application

This section describes all the steps necessary for migrating the application from the source environment to the WebSphere environment. The steps involve the creation of the corresponding database in DB2 Universal Database V8.2 since the source environment runs with HSQL. Once the database is defined, we need to configure Hibernate to use DB2.

The migration steps for this applications include some code changes. This section also describes how to change the code, and how we found and fixed the problems we encountered.

Finally, we need to configure the WebSphere Application Server resources to deploy the application and, once they are configured, deploy and test the application.

#### Configuring DB2 Universal Database V8.2

The current version of ivata groupware supports MySql, HSQL and has scripts generated by Apache Torque for a number of different databases, including Oracle and DB2, although none of those scripts is tested, except for MySql and HSQL.

This section describes how to create and populate ivata groupware database to DB2 Universal Database V8.2. The application uses the Apache Torque project to generate schema DDL (Data Definition Language) and sample data SQL for different databases. Apache Torque allows you to specify a database schema in an XML file and then generate a database-specific DDL from that configuration. It can also generate POJOs (Plain Old Java Object) for that schema and serve as persistence framework, but ivata groupware is not using the latter part of Apache Torque since it uses Hibernate instead.

The DDL file that comes with the ivata groupware download works well to create schema in DB2, but the generated SQL to populate the database with sample data is not correct. We had to change the order of SQL statements to make it work. The original file had Primary Key conflicts due to the wrong order of SQL statements. The proper SQL file named `ivata_db2_data.sql` is available for download with the additional material for this book. Refer to Appendix B, “Additional material” on page 323 for further details.

Now you need to create the application database and tables. Assuming that the DB2 is up and running from a command line window, type the following commands:

```
cd <source_home>\package\install\src\db\db2
db2cmd
```

This command will open a new command line window with a DB2 Command Window environment set. Once you are in this new DB2 Command Window, you need to run the following DB2 commands to create and populate the database:

```
db2 create db ivataDB
db2 connect to ivataDB user db2admin using its0ra10
db2 -tvf schema-current.sql
db2 -tvf ivata_db2_data.sql
```

**Note:** Make sure you use the `ivata_db2_data.sql` file provided with the additional materials for this book and not the file `data-current.sql` that comes with the ivata groupware 0.11 download.

## Configuring Hibernate

The original version of ivata groupware is using HSQL to store all its data. Because we wanted to use a production grade database, we switched it to DB2. To do so, you need to update the Hibernate configuration file as shown in Example 8-1 on page 269. The `hibernate.cfg.xml` file is located in the following directory:

```
<source_home>\hibernate\src\xml\hibernate.cfg.xml
```

### Example 8-1 Hibernate configuration for DB2 JDBC driver

---

```
<hibernate-configuration>
  <session-factory>
    . . .
    <!--      DB STUFF - for DB2      -->
    <property name="dialect">net.sf.hibernate.dialect.DB2Dialect</property>
    <property name="connection.driver_class">
      COM.ibm.db2.jdbc.app.DB2Driver
    </property>
    <property name="hibernate.connection.url">jdbc:db2:ivatab</property>
    <property name="hibernate.connection.username">db2admin</property>
    <property name="hibernate.connection.password">its0ra10</property>
    <property name="show_sql">>false</property>
    . . .
  </session-factory>
```

---

Before changing any configuration, we made sure the application worked with HSQL first, then configured the application to use DB2. The next natural step is to verify that the application works with DB2 and thus you will need to rebuild it and redeploy into Tomcat. To rebuild and redeploy the application into Tomcat, follow the steps described in “Building the ivata groupware application” on page 264 and “Deploying the application” on page 265.

Once the application is redeployed, test it as described in “Testing the application” on page 266. Now that you know the application works in Tomcat with DB2, you can migrate it to WebSphere Application Server V6.

### Updating the JSP source code

During our migration, we wanted to see if we could deploy the ivatagroupware-war-0.11.1.war file directly into WebSphere Application Server V6 without making any modifications (exactly the same war file we deployed into Tomcat). The deployment worked, but when we logged in to the application, we saw errors on the left part of the window, as shown in Example 8-2.

### Example 8-2 Error message during initial deployment of ivata groupware

---

JSP Processing Error  
HTTP Error Code: 500

Error Message:JSPG0048E: Page failed to validate using taglib validator for /WEB-INF/tag/jstl/c-rt.tld : <p>org.xml.sax.SAXParseException: Attribute name "jsp:id" associated with an element type "**c:if**" must be followed by the ' = ' character.</p>

Root Cause:com.ibm.ws.jsp.translator.JspTranslationException: JSPG0048E: Page failed to validate using taglib validator for /WEB-INF/tag/jstl/c-rt.tld : org.xml.sax.SAXParseException:

```

Attribute name "jsp:id" associated with an element type "c:if" must be followed by the ' = '
character.at
com.ibm.ws.jsp.translator.visitor.validator.ValidateVisitor.validateTagLib(ValidateVisitor.java
:956)at
com.ibm.ws.jsp.translator.visitor.validator.ValidateVisitor.visitJspRootStart(ValidateVisitor.j
ava:453)at com.ibm.ws.jsp.translator.visitor.JspVisitor.processJspElement(JspVisitor.java:124)
at com.ibm.ws.jsp.translator.visitor.JspVisitor.visit(JspVisitor.java:110)at
com.ibm.ws.jsp.translator.JspTranslator.processVisitors(JspTranslator.java:121)at
com.ibm.ws.jsp.translator.utils.JspTranslatorUtil.translateJsp(JspTranslatorUtil.java:168)at
com.ibm.ws.jsp.translator.utils.JspTranslatorUtil.translateJspAndCompile(JspTranslatorUtil.java
:81)at
com.ibm.ws.jsp.webcontainerext.JSPExtensionServletWrapper.translateJsp(JSPExtensionServletWrapp
er.java:319)at
com.ibm.ws.jsp.webcontainerext.JSPExtensionServletWrapper._checkForTranslation(JSPExtensionServ
letWrapper.java:292)at
com.ibm.ws.jsp.webcontainerext.JSPExtensionServletWrapper.checkForTranslation(JSPExtensionServl
etWrapper.java:234)at
com.ibm.ws.jsp.webcontainerext.JSPExtensionServletWrapper.handleRequest(JSPExtensionServletWrap
per.java:139)at com.ibm.ws.webcontainer.webapp.WebApp.handleRequest(WebApp.java:2872)at
com.ibm.ws.webcontainer.webapp.WebGroup.handleRequest(WebGroup.java:220)at
com.ibm.ws.webcontainer.VirtualHost.handleRequest(VirtualHost.java:204)at
com.ibm.ws.webcontainer.WebContainer.handleRequest(WebContainer.java:1779)at
com.ibm.ws.webcontainer.channel.WCChannelLink.ready(WCChannelLink.java:77)at
com.ibm.ws.http.channel.inbound.impl.HttpInboundLink.handleDiscrimination(HttpInboundLink.java:
466)at
com.ibm.ws.http.channel.inbound.impl.HttpInboundLink.handleNewInformation(HttpInboundLink.java:
405)at
com.ibm.ws.http.channel.inbound.impl.HttpICLReadCallback.complete(HttpICLReadCallback.java:104)
at com.ibm.ws.tcp.channel.impl.WorkQueueManager.requestComplete(WorkQueueManager.java:555)at
com.ibm.ws.tcp.channel.impl.WorkQueueManager.attemptIO(WorkQueueManager.java:608)at
com.ibm.ws.tcp.channel.impl.WorkQueueManager.workerRun(WorkQueueManager.java:941)at
com.ibm.ws.tcp.channel.impl.WorkQueueManager$Worker.run(WorkQueueManager.java:1028)at
com.ibm.ws.util.ThreadPool$Worker.run(ThreadPool.java:1394)

```

---

As it is evident from the exception, this error is caused by one or more of the JSTL `<c:if>` tags that is not coded properly. The JSTL tag `<c:if>` makes it possible to dynamically generate template text on one condition. The `<c:if>` tag generates its body if the expression in the test attribute evaluates to the boolean value `true` or the string value `"true"`.

This error came from the `left.jsp` file. It had eight `<c:if>` tags. For problem determination, we decided to comment out all of these tags and rerun the application. This time it worked well (no compilation errors). This way, we found out that we just needed to fix the `<c:if>` tags and change the Hibernate connection pool to use the WebSphere datasource.

**Important:** We took the war file that was built by Maven to be deployed into Tomcat and without making any modifications, we were able to deploy and run it in WebSphere. This is a Web application and does not use all of the J2EE features, but it is using a large number of open source frameworks, such as Hibernate, Struts, Apache Commons or OpenSymphony OSCache, which makes it interesting from the migration point of view. Ease of portability is a great strength of J2EE.

### ***Problem determination and resolution***

To fix the error with the `<c:if>` tag, we used a simple text editor such as Notepad and opened the following file:

```
<source_home>\package\war\src\web\left.jsp
```

We commented out all occurrences of the `<c:if>` tag as shown in the following examples.

#### ***Example 8-3 Original left.jsp file***

---

```
<c:if test='<%=menuForSize.getId().equals(initialId)%>'>
    <%visibleMenu = idCount;%>
</c:if>
```

---

#### ***Example 8-4 left.jsp after changes***

---

```
<!-- c:if test='<%=menuForSize.getId().equals(initialId)%>' -->
    <%visibleMenu = idCount;%>
<!-- /c:if -->
```

---

**Note:** You do not have to follow all the steps of our problem determination and solving. We are showing all the steps for the sake of explaining the method we used to find and solve the errors.

There is a total of eight tags in left.jsp. We saved the file with our changes and rebuilt the war file using Maven. Then we redeployed the application into WebSphere and this time saw no errors. We will show you deployment steps later in “Deploying the application” on page 278.

We uncommented those `<c:if>` tags one by one and tried to run the application. We used the deployment directory in IvataProfile to edit the file directly so that we did not have to redeploy the application every time. Once we changed and saved the JSP file, it was reloaded by WebSphere automatically.

What we found is that in two places, those tags were not coded properly. Thus, we had to correct them. Here is the original version of the line 489 in the left.jsp file:

```
<c:if test='<%=link.indexOf(':') == -1) && !link.startsWith("/")%>'>
```

And here is corrected version:

```
<c:if test='<%=link.indexOf(":") == -1) && !link.startsWith("/")%>'>
```

Note the use of double quotes (":") instead of single quotes (':'). It is not surprising that these tags did not work. Since there is a Java expression inside of those tags, we have to use double quotes for string literals in Java. We also have to update line 521 and fix the same error.

You may want to review the JSTL documentation available at the following URLs for further details.

<http://java.sun.com/products/jsp/jstl/index.jsp>

<http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html>

## Changing the Hibernate connection pool to WebSphere datasource

When we performed the migration, we moved in small steps. We changed the minimum number of properties before moving forward to the next step. We tried to make sure that the baseline worked well (the application runs in Tomcat) and then we tried to deploy the same configuration to WebSphere.

This is a useful technique to isolate changes and have a more controlled environment. However, in this book, we are not taking you through all the steps in the exact order we used (we simplified the process and omitted the trial and error). Instead, we show you how to make all changes first and then build and deploy the application in one step. Usually, this will be very iterative process, but to simplify the description, we are not showing all intermediate deployments and builds that we had to perform.

Considering that the application works with DB2 using the Hibernate connection pool, you need to change the connection pool mechanism, since Hibernate's own connection pooling algorithm is quite rudimentary. It is not designed for production use and so you have to use the WebSphere-provided connection pool as it is documented on the Hibernate site at this URL:

[http://www.hibernate.org/hib\\_docs/reference/en/html/session-configuration.html](http://www.hibernate.org/hib_docs/reference/en/html/session-configuration.html)

There are many benefits to using the WebSphere connection pool implementation and Hibernate can take advantage of those automatically. Hibernate JDBC connections obtained via JNDI from the WebSphere datasource



will automatically participate in the container-managed transactions of the application server.

In order to implement these changes, we need to update the Hibernate configuration file located in the following directory, as described in Example 8-5.

<source\_home>\hibernate\src\xml\hibernate.cfg.xml

*Example 8-5 Define datasource in hibernate.cfg.xml*

---

```
<hibernate-configuration>
  <session-factory>
    . . .
    <!--      DB STUFF - for DB2      -->
    <property name="connection.datasource">
      java:comp/env/jdbc/ivataDS
    </property>
    <property name="dialect">net.sf.hibernate.dialect.DB2Dialect</property>
    <property name="show_sql">false</property>
    . . .
  </session-factory>
</hibernate-configuration>
```

---

**Note:** Note that we have only three properties defined here. These are the only properties needed by Hibernate to get connected to the database.

When developing an application, you generally do not know about the name of the datasource on the target application server. In your code, you do not look up the datasource directly. Instead, you look up the resource reference from the `java:comp/env` namespace file. In order to bind the datasource from the Web application to the WebSphere runtime, you need to specify the resource reference in the following file, as described in Example 8-6.

<source\_home>\package\war\src\web\WEB-INF\web.xml

*Example 8-6 web.xml file for ivata groupware application*

---

```
<web-app>
  . . .
  <error-page>
    . . .
  </error-page>

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

---

```
<env-entry>
    . . .
</env-entry>
    . . .
</web-app>
```

---

Once we made all the changes to the configuration just mentioned and rebuilt the application, we redeployed the application to WebSphere. This time, we saw an HTTP 500 error message in the browser:

```
Error 500: Exception forwarding for name loginGuestAction:
javax.servlet.UnavailableException: Cannot initialize RequestProcessor of class
com.ivata.groupware.container.struts.PicoRequestProcessor:
java.lang.NullPointerException
```

To see more details on this error message, we reviewed the output from the ivata groupware application. This error log file can be found in the following directory:

```
<was_home>\profiles\<profile_name>\logs\server1\SystemOut.log
```

---

*Example 8-7 SystemOut.log file from ivata groupware application*

---

```
[5/23/05 17:56:07:563 EDT] 00000036 SystemOut      0 4466 [WebContainer : 0] ERROR
com.ivata.groupware.container.PicoContainerFactory - java.lang.NullPointerException thrown
looking for class called 'null'
java.lang.NullPointerException
    at java.lang.Class.forName1(Native Method)
    at java.lang.Class.forName(Class.java:Compiled Code))
    at
com.ivata.groupware.container.PicoContainerFactory.className(PicoContainerFactory.java:172)
    at
com.ivata.groupware.container.PicoContainerFactory.initializeSettingsCache(PicoContainerFactory
.java:416)
    at
gjdk.com.ivata.groupware.container.PicoContainerFactory_GroovyReflector.invoke(PicoContainerFac
tory_GroovyReflector.java)
    . . .
    at com.ibm.ws.tcp.channel.impl.WorkQueueManager.requestComplete(WorkQueueManager.java:555)
    at com.ibm.ws.tcp.channel.impl.WorkQueueManager.attemptIO(WorkQueueManager.java:608)
    at com.ibm.ws.tcp.channel.impl.WorkQueueManager.workerRun(WorkQueueManager.java:941)
    at com.ibm.ws.tcp.channel.impl.WorkQueueManager$Worker.run(WorkQueueManager.java:1028)
    at com.ibm.ws.util.ThreadPool$Worker.run(ThreadPool.java:Compiled Code))
```

---

We noticed that the error originated from line 416 in the PicoContainerFactory.java file in the following directory:

```
<source_home>\core\src\java\com\ivata\groupware\container\
```

By looking at that file, we realized that the Groovy initialization script for the PicoContainer looks up the Hibernate properties file and uses the JDBC driver information to access the database directly without using Hibernate for persistence. This is only done at initialization time when Hibernate is not yet loaded.

The NullPointerException exception shown in Example 8-7 on page 274 occurs because we removed the driver and URL information from the Hibernate file. Normally, when you use the datasource to access your database, you do not need to provide the driver, user, password and URL information in the Hibernate configuration file. But in this case, we need to have both, therefore we had to change the hibernate.cfg.xml file to make it look like the following example.

*Example 8-8 Define datasource in hibernate.cfg.xml*

---

```
<hibernate-configuration>
  <session-factory>
    . . .
    <!--      DB STUFF - for DB2      -->
    <property name="connection.datasource">
      java:comp/env/jdbc/ivataDS
    </property>
    <property name="dialect">net.sf.hibernate.dialect.DB2Dialect</property>
    <property name="show_sql">false</property>
    <property name="connection.driver_class">
      COM.ibm.db2.jdbc.app.DB2Driver
    </property>
    <property name="hibernate.connection.url">jdbc:db2:ivatadb</property>
    <property name="hibernate.connection.username">db2admin</property>
    <property name="hibernate.connection.password">its0ral0</property>
    . . .
  </session-factory>
</hibernate-configuration>
```

---

## Building the application

With all these modifications in place, you are now ready to build the ivata groupware application so that you have a war file ready for deployment to WebSphere Application Server V6.

From the <source\_home> directory, run this command:

```
maven
```

The resulting war file is located in the following directory:

```
<source_home>\package\war\target\ivatagroupware-war-0.11.1.war
```

At this point, you need to configure a WebSphere datasource in order to deploy the application.

## Creating a new datasource

Follow these steps to configure ivata groupware to use the WebSphere Application Server V6 datasource. We assume that you have the application server server1 up and running.

**Note:** For each migration exercise, we created a new WebSphere profile so that we would have a clean environment for deploying the migrated applications. The creation on WebSphere profiles is optional.

1. Open the WebSphere Administrative Console:  
`http://localhost:9060/ibm/console`
2. Log in and click **Resources** → **JDBC Providers** and then **New** to create a new JDBC provider with the following characteristics.

Table 8-1 JDBC provider values

Parameter	Value
Database type	DB2
Provider type	DB2 Legacy CLI-based Type 2 JDBC Driver
Implementation type	XA datasource

3. Click **Next**. Specify the correct path to the DB2 JDBC driver and click **Apply**. In our migration example, the path to the JDBC driver is:  
`C:\IBM\SQLLIB\java\db2java.zip`
4. Next, click **Data sources** in the Additional properties section and create a new datasource with the following characteristics.

Table 8-2 Datasource values

Parameter	Value
Name	ivataDS
JNDI Name	jdbc/ivataDatasource
Database Name	ivatadb

5. Click **Apply**.

6. Next, click **J2EE Connector Architecture (J2C) authentication data entries** in the Related items section.
7. Create a new alias and specify the following values.

Table 8-3 J2C alias configuration

Parameter	Value
Alias	DB2user
UserId	db2admin
Password	its0ral0

8. Click **OK** and go back to the datasource you just created. From the *Component-managed authentication alias* drop-down menu, select the J2C alias you just created.
9. Click **OK** and save your configuration.

Next, verify the connection to the database. Select the datasource you just created from the list and click **Test connection**.

### Configuring the shared library for the Direct JDBC access

Since the ivata groupware application uses direct JDBC access, in addition to Hibernate, you need to add the DB2 JDBC driver to the application classpath. We recommend that when you migrate your application, you rewrite your code to use the WebSphere datasource and avoid using the JDBC driver directly. If you rewrite your application, you will not have to perform this step. It is not good practice to use the JDBC driver directly since it does not take advantage of the WebSphere-provided qualities of services, advanced connection pooling capabilities and transaction management.

To define the JDBC driver, we need to follow these steps:

1. From the WebSphere Administrative Console, click **Environment** → **Shared Libraries** → **New** to specify a container-wide shared library with the following characteristics.

Table 8-4 Shared library values

Parameter	Value
Name	DB2driver
Classpath	<db2_home>java\db2java.zip

2. Click **OK** and save the configuration.

## Deploying the application

There are several different ways to deploy applications and create resources (datasources, etc.) in WebSphere Application Server V6. Usually, you would use the WebSphere Administrative Console to perform these tasks for the first time, but for production environments, you would normally write a script with the wsadmin tool instead.

For a development environment, you could automatically deploy applications from Rational Application Developer V6 with a single click. Since we are not using the Rational product, we performed a manual deployment. If we had to deploy the application multiple times (as in the real world), we could use a wsadmin script and make it part of our build process. For more information about different ways of deploying applications, visit the following URL:

[http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/crun\\_app\\_install.html](http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/crun_app_install.html)

In this case, we only have one application, and all the necessary resources are already configured in the WebSphere server1 instance, so we can deploy the application through the WebSphere Administrative Console, following these steps:

1. From the WebSphere Administrative Console, click **Applications** → **Install New Application** in the navigation pane to the left.
2. Specify the path to the EAR file you just created and the context root as described in the following table and then click **Next**.

Table 8-5 Installation values

Parameter	Value
Local File System Path	<source_home>\package\war\target\ivatagroupware-war-0.11.1.war
Context Root	/ivata

3. Select **Generate Default Bindings** on the next page. Click **Next**.
4. A new window appears with Application Security Warnings. Click **Continue**.
5. In the *Step 1: Select installation options* window, leave all the default options and click **Next**.
6. In the *Step 2: Map modules to servers* window, leave all the default options and click **Next**.
7. In the *Step 3: Map resource references to resources* window, select the **jdbc/ivataDataSource** datasource in the drop-down box at the top, then check the box next to the **ivatagroupware-war-0.11.1.war** module and click

**Apply** next to the datasource. Then select the application module again, select the **DB2user** authentication entry that you created previously and click **Apply**, then click **Next**.

8. You may receive an Application Resource Warnings window. Click **Continue**.
9. In the *Step 4: Map virtual hosts for Web modules* window, click **Next**.
10. In the *Step 5: Summary* window, click **Finish**. It may take a minute to deploy the application and validate all mappings. Once you are finished, save the configuration.

At this point, you have deployed your application. Since ivata groupware uses direct JDBC access in addition to the datasource, you need to configure one last thing before starting the application.

1. From the WebSphere Administrative Console, click **Applications** → **Enterprise Applications** → **ivatagroupware-war-0\_11\_1\_war**.
2. In the Additional Properties section, click **Libraries**. Click **Add**, then select the **DB2driver** library that you have defined earlier.
3. Click **OK** and save the configuration.

Now you need to start the application.

1. From the WebSphere Administrative Console, click **Applications** → **Enterprise Applications**, then select the checkbox to the left of the ivata groupware application and click **Start**.

You should see the application started and a green arrow to the right of the application name, as shown in Figure 8-1.

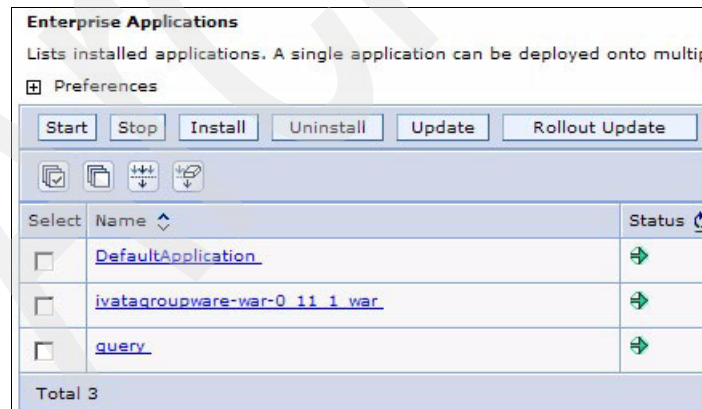


Figure 8-1 ivata groupware deployed and started

2. Log in to the application using the following URL:

`http://localhost:9080/ivata/index.jsp`

**Note:** We are using the default HTTP transport port. If you are using another profile, you have to use the corresponding port. Refer to 4.4.3, “Creating new WebSphere profiles” on page 75 for more information about profiles and their ports.

## Testing the application

Now you are ready to test the application the same way you tested it when it was deployed to Tomcat.

Browse through the application to verify its functionality. We logged in the application, selected the calendar option and added some events on different days. We browsed all the events using the arrows in the calendar, and selected one to modify its content.

We browsed through the contact list and added a contact, then we clicked the icons to maintain address books and maintain groups in address books. After that simple test, we checked the log files and did not find any errors.

Once again, this is not a complete test, and it is not recommended for a production environment, but it was good enough to know that the application was successfully migrated.

## 8.5 xPetstore Servlet migration

xPetstore Servlet is an LGPL licensed open source project hosted by SourceForge. It is a complete rewrite of Sun's PetStore J2EE reference application. xPetstore Servlet is a J2EE 1.3 application that demonstrates the capabilities of the J2EE platform.

xPetstore comes in two versions. One is based on EJBs and the other is based on Servlets and Hibernate. The xPetstore EJB version migration is covered in Chapter 6, “Migrating from BEA WebLogic” on page 111.

In this example, we will migrate the servlet version of xPetstore since Tomcat is not a full-blown application server.

xPetstore Servlet was included in this book for the following reasons:

- ▶ It is based on popular open source frameworks.
- ▶ It was designed to run on multiple application servers.
- ▶ It is not specifically designed to run on WebSphere Application Server V6.



xPetstore is an interesting migration example because it was implemented with several technologies and frameworks, as described in Table 1-2 on page 4.

### 8.5.1 Migration approach

As discussed in 3.4, “Migration alternatives” on page 51, there are different alternatives to migrate this application. For the xPetstore Servlet migration, we selected the method described in 3.4.1, “Alternative 1: Keeping the existing development environment” on page 52.

We used the original development environment with xDoclet and Ant. Instead of Rational Application Developer V6 or any other IDE, we used a plain text editor (such as jEdit or another editor with color syntax highlight).

We wanted to show that it is possible to migrate an application like xPetstore to WebSphere Application Server V6 without using an IDE and that it is possible to maintain the original development environment.

### 8.5.2 Configuring the source environment

This section describes all the required steps for setting up the source environment. Follow the instructions in 8.3, “Software installation” on page 260 to install the software needed in the source environment.

xPetstore was designed to run on Tomcat 4.x, so we need to perform some extra steps to migrate it to Apache Tomcat 5.5.9. This section also describes how we migrated to DB2 Universal Database V8.2.

#### Getting the xPetstore Servlet application

Before starting the migration, you need to obtain source files for the xPetstore Servlet application. You can download file xpetstore-3.1.3.zip with all source files from the following URL:

<http://xpetstore.sourceforge.net>

In our scenario, we downloaded and extracted the application to the following directory (later referred to as <source\_home>):

D:\sampleApp\xpetstore-tomcat

#### Installing and configuring the prerequisites

xPetstore Servlet uses the JavaMail API, but this is no longer bundled with Apache Tomcat 5.5.9. So it is necessary to manually install JavaMail and its dependencies. Follow these steps to install it:

1. Download the JavaMail API from the following URL:

<http://java.sun.com/products/javamail>

2. Download the JavaBeans Activation Framework, on which JavaMail depends, from the following URL:

<http://java.sun.com/products/javabeans/glasgow/jaf.html>

3. Copy the mail.jar file from the JavaMail installation package to the following directory:

<tomcat\_home>\common\lib

4. Copy the activation.jar file from the JavaBeans Activation Framework installation package to the following directory:

<tomcat\_home>\common\lib

## Configuring the database

This section describes how to configure the xPetstore application to use DB2 Universal Database V8.2 since xPetstore does not support DB2 by default.

In order to use DB2 Universal Database V8.2, you have to make some changes in the configuration files of the application. These changes are necessary because the application is using Hibernate to map the application objects with the relational data, and it needs to know which database it is accessing, and some other configuration data, such as user names and passwords.

Follow these steps to configure xPetstore Servlet to use DB2 Universal Database V8.2.

1. Edit the following file:

<source\_home>\conf\db\database.properties

and change the database property to:

database=db2

2. In the same directory, create a file called db2.properties like the one shown in Example 8-9.

### *Example 8-9 xPetstore database configuration file*

---

```
#=====
# DB2 configuration
#=====
db.driver=COM.ibm.db2.jdbc.app.DB2Driver
db.url=jdbc:db2:to_xp_se
db.user=db2admin
db.password=its0ra10
db.classpath=${lib.dir}/db2java.zip
db.foreign.key=true
```

```
hibernate.dialect=cirrus.hibernate.sql.DB2Dialect
hibernate.generator.class=native
hibernate.outer.join=true
```

---

3. Run the **ant** command from the following directory:

```
<source_home>\xpetstore-servlet
```

This will create the DDL for DB2 Universal Database V8.2. The file's location is:

```
<source_home>\xpetstore-servlet\build\sql\schemas.sql
```

The database mapping for Hibernate is created with the Ant `xdoclet.hibernate.ddl` target.

**Note:** You will need to run Ant again later to build the final war. Running Ant more than once during our example is done with the sole purpose of making the steps more clear and having the migration process separated into smaller modules.

4. Create the database by running the DDL script generated by Hibernate from the previous step. This is done by executing the following commands from a console.

```
db2cmd
db2 create db to_xp_se
db2 connect to to_xp_se user db2admin using its0ra10
db2 -tvf <source_home>\xpetstore-servlet\build\sql\schemas.sql
db2 -tvf <source_home>\xpetstore-servlet\sql\data.sql
```

## Configure the datasource

Tomcat 5.5 datasources can be configured either in the `<tomcat_home>\conf\server.xml` file or in a configuration file located within the Web application itself. In this example, we use the second option, since it is a Tomcat-specific feature. This feature is available in Tomcat 5.5.4 and later versions.

Follow these steps to set up the datasource:

1. Create a new directory called META-INF in the following directory:

```
<source_home>\xpetstore-servlet\web\
```

2. Create a new file called `context.xml` inside of the META-INF directory, containing the configuration as shown in Example 8-10 on page 284.

*Example 8-10 Apache Tomcat 5.5.9 datasource configuration file for xPetstore*

---

```
<Context reloadable="true">
  <Resource
    auth="Container"
    description="DB2 DataSource"
    name="jdbc/xpetstoreDS"
    type="javax.sql.DataSource"
    driverClassName="COM.ibm.db2.jdbc.app.DB2Driver"
    maxIdle="2"
    maxWait="5000"
    username="db2admin"
    password="its0ra10"
    url="jdbc:db2:to_xp_se"
    maxActive="4"/>

  <Resource name="mail/xpetstore/MailSession"
    auth="Container" type="javax.mail.Session"/>
  <ResourceParams name="mail/xpetstore/MailSession">
    <parameter>
      <name>mail.smtp.host</name>
      <value>localhost</value>
    </parameter>
  </ResourceParams>

  <!-- Monitored resources -->
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <WatchedResource>META-INF/context.xml</WatchedResource>
</Context>
```

---

3. Modify the build script, located in <source\_home>\xpetstore-servlet\build.xml, to include context.xml in the war file. This is done by inserting the following tag inside the war tag in the target named war.webwork, as highlighted in Example 8-11.

*Example 8-11 Build script modification needed to include the context.xml file*

---

```
<project name="xpetstore-servlet" default="all" basedir=".">
...
  <target name="war.webwork" depends="init">
    ...
    <war
      destFile="${dist.dir}/${war.name}"
      webxml="${build.dir}/WEB-INF/web.xml"
    >
      ...
      <metainf dir="${web.dir}/META-INF">
        <include name="**.*" />
      </metainf>
    </war>
  </target>
</project>
```

```
</war>
</target>
...
```

---

## Building the xPetstore Servlet application

At this point, you have made all the necessary changes to run xPetstore using DB2 Universal Database V8.2 and running under Apache Tomcat 5.5.9.

xPetstore uses Ant build scripts to build the application. To make a build execute **ant** in the following directory:

```
<source_home>\xpetstore-servlet
```

The build script will create the following war file:

```
<source_home>\dist\xpetstore-servlet.war
```

This file contains all the information, including the datasource definition, needed by Tomcat to deploy the application.

## Deploying the application

Tomcat will automatically deploy Web applications located in the webapps folder by default. Follow these steps to deploy xPetstore in Tomcat.

1. Copy the war file from:

```
<source_home>\dist\xpetstore-servlet.war
```

to:

```
<tomcat_home>\webapps
```

2. Open a command line and change the directory to:

```
<tomcat_home>\bin
```

3. Start Tomcat by issuing the following command:

```
startup
```

You should see no errors in the log if the application was deployed successfully.

4. Access the application at the following URL:

```
http://localhost:8080/xpetstore-servlet/
```

## Testing the application

The servlet version of xPetstore has no automated tests implemented with JUnit, JUnitEE, HttpUnit or any other automated test tool. Therefore, we have to test the application manually.

The test is done by performing the use case actions described at the following URL:

<http://xpetstore.sourceforge.net/specifications.html>

In a simple test, we browsed the application, signed in using user1 and password1, navigated the different animals and races and added one to the cart. We reviewed the cart items and checked out. This simple procedure took us through the application with no errors. We checked the log files and did not find any error or warnings.

At this point, we have successfully built, deployed and tested xPetstore Servlet into Apache Tomcat 5.5.9, which is its original environment. Having the application working properly on the source environment is our starting point for migrating into WebSphere Application Server V6.

### 8.5.3 Migrating the xPetstore Servlet application

This section describes all the necessary steps for migrating the application from Tomcat to WebSphere Application Server. The steps of this migration involve only configuration files, so you will not have to change the application source code.

In this section, you will find how to modify the build files and customize them to prepare the application to run in WebSphere Application Server V6. Like we did in the ivata groupware migration earlier in this chapter, we also have to modify the Hibernate configuration files to make them run with DB2 Universal Database V8.2.

Once you have all those steps completed, the application will be ready to be built and deployed in WebSphere Application Server V6.

The last part of this section describes how to configure the resources required by the xPetstore Servlet to run in WebSphere Application Server V6, and how to deploy and test the migrated application.

#### Build script

Some xPetstore configurations are specific to the application server or database you are using. You need to configure the Ant build script provided with the application source to build the application to run in WebSphere Application Server V6.

To modify the build script, follow these steps:

1. Edit the following file:

```
<source_home>\conf\as\appserver.properties
```

Set the app.server property to WebSphere:

```
app.server=websphere
```

The resulting file should look like Example 8-12.

*Example 8-12 appserver.properties configured to use WebSphere*

---

```
=====
# Application server configuration file
=====

# Select the application server:
# - jboss
# - weblogic
# - orion
# - tomcat (For xpetstore-servlet only)
app.server=websphere

# logical JNDI of the javax.jms.QueueConnectionFactory
jndi.queue.ConnectionFactory=jms/xpetstore/QueueConnectionFactory

# logical JNDI of the queue used by xpetstore.services.order.OrderProcessorMDB
jndi.queue.order=jms/queue/xpetstore/order

# logical JNDI of the queue used by xpetstore.services.mail.MailerMDB
jndi.queue.mail=jms/queue/xpetstore/mail

# logical JNDI of javax.mail.MailSession
jndi.mail.session=mail/xpetstore/MailSession

# logical JNDI of javax.sql.DataSource
jndi.datasource=jdbc/xpetstore
```

---

2. Create a new WebSphere configuration file in the following location:

```
<source_home>\conf\as\websphere.properties
```

and copy into that file the content of the following example.

*Example 8-13 WebSphere configuration file*

---

```
=====
# WebSphere configuration file
=====

# physical JNDI of the datasource
websphere.datasource=java:comp/env/jdbc/xPetstoreDS

# physical JNDI name of the javax.jms.MailSession
websphere.mail.session=mail/xpetstore/MailSession
```

```
# Datasource used by Hibernate
# DO NOT CHANGE THIS
hibernate.datasource=${websphere.datasource}
```

---

**Note:** If you pay attention to the values of the datasource, you will see that the websphere.datasource property has the following value:

```
websphere.datasource=java:comp/env/jdbc/xPetstoreDS
```

while in Tomcat the same configuration parameter looks like this:

```
tomcat.datasource=java:/comp/env/jdbc/xPetstoreDS
```

This difference is important because you may be accustomed to using the Tomcat form, and this kind of error is very difficult to find.

3. Remove the Tomcat specific configuration file located in:

```
<source_home>\xpetstore-servlet\web\META-INF\context.xml
```

4. Modify the build script so that duplicate files are not allowed in the WAR file.

The xPetstore build script creates a WAR file that contains duplicate files. The WAR file runs on Tomcat but will generate an error on WebSphere Application Server V6 as illustrated in Example 8-14.

*Example 8-14 Excerpt from the WebSphere Application Server log*

---

```
Duplicate files in WAR. [10.5.2005 21:38:59:231 EEST] 00000032 SystemErr      R
java.util.zip.ZipException: duplicate entry: WEB-INF/classes/webwork.properties
    at java.util.zip.ZipOutputStream.putNextEntry(ZipOutputStream.java:185)
```

---

To remove duplicates from the WAR file, change the war.webwork target in the build.xml file located in the following directory:

```
<source_home>\xpetstore-servlet\build.xml
```

so that it looks as shown in Example 8-15.

*Example 8-15 build.xml configuration to avoid duplicated files*

---

```
<project name="xpetstore-servlet" default="all" basedir="."/>
...
<target name="war.webwork" depends="init">
  <delete>
    <fileset dir="${dist.dir}" includes="${war.name}" />
  </delete>
  <war
    destFile="${dist.dir}/${war.name}"
    webxml="${build.dir}/WEB-INF/web.xml"
  />
</target>
```



```

        duplicate="preserve"
    >
    ...
</war>
</target>
...

```

---

## Configuring the datasource

Follow these steps to configure xPetstore to use the WebSphere Application Server V6 datasource.

1. Add a resource reference to the datasource in the web.xml file. Since XDoclet generates the web.xml, you have to tell XDoclet to add the resource reference. In our migration example, this is done by creating the following XDoclet merge file:

```
<source_home>\xpetstore-servlet\xdoclet\web\web-resource-env-refs.xml
```

2. Copy into the merge file you just created the content of Example 8-16.

*Example 8-16 Resource reference configuration to be used in web.xml*

---

```

<resource-ref id="ResourceRef_1115821976066">
  <description>
  </description>
  <res-ref-name>jdbc/xPetstoreDS</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>

```

---

XDoclet will merge the file above with the web.xml that is generated dynamically.

**Note:** WebSphere Application Server V6 allows you to map the resource reference specified in web.xml to a JNDI name when the application is deployed. You can also use a WebSphere-specific deployment descriptor to map the resource reference to the JNDI name.

Using the datasource's JNDI name directly will generate the following error message.

*Example 8-17 Excerpt from the WebSphere log*

---

```

[11.5.2005 23:21:19:004 EEST] 0000011b NamingHelper I    JNDI InitialContext
properties:{}

```

```
[11.5.2005 23:21:19:044 EEST] 0000011b ConnectionFac W J2CA0294W: Deprecated
usage of direct JNDI lookup of resource jdbc/xPetstoreDS. The following
default values are used: [Resource-ref settings]
```

```
    res-auth:                1 (APPLICATION)
    res-isolation-level:     0 (TRANSACTION_NONE)
    res-sharing-scope:       true (SHAREABLE)
    loginConfigurationName:  null
    loginConfigProperties:   null
[Other attributes]
```

```
    res-resolution-control:  999 (undefined)
isCMP1_x:                   false (not CMP1.x)
isJMS:                      false (not JMS)
```

```
[11.5.2005 23:21:19:124 EEST] 0000011b DatasourceCon I Using datasource:
jdbc/xPetstoreDS
```

---

## Creating the Log4j configuration file

xPetstore does not provide the Log4j configuration file. Starting the application without it will result in the following error.

### *Example 8-18 Log4j error message*

---

```
[10.5.2005 22:11:11:790 EEST] 00000034 SystemErr      R log4j:WARN No appenders
could be found for logger (xpetstore.web.servlet.ActionServlet).
```

```
[10.5.2005 22:11:11:790 EEST] 00000034 SystemErr      R log4j:WARN Please
initialize the log4j system properly.
```

---

In order to avoid this error, you have to create a new directory called *classes* inside of <source\_home>\xpetstore-servlet\web\WEB-INF, create a log4j.properties file in that directory and copy the content of Example 8-19 into that file.

### *Example 8-19 Log4j configuration file*

---

```
log4j.rootCategory=DEBUG, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{DATE} %p [%t] %C{1}.%M(%L) |
%m%n
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=logs/xpetstore.log
log4j.appender.R.MaxFileSize=2048KB
```

```
log4j.appender.R.MaxBackupIndex=5
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern= %d{DATE} %p [%t] %C{1}:%M(%L) | %m%n
```

---

## Building the application

All the changes needed to deploy the application in WebSphere Application Server are complete. Even though you have just modified the configuration files, you still need to run the build command to create the new war file.

To make a build, execute the **ant** command in the following directory:

```
<source_home>\xpetstore-servlet
```

The build script will create the following WAR file:

```
<source_home>\dist\xpetstore-servlet.war
```

This is the file you need to deploy in WebSphere Application Server V6.

## Configuring the Mail Provider in WebSphere

The JavaMail session provides an abstraction layer for communicating with an e-mail server. This section describes how to configure a Mail Provider which is used to create a JavaMail session in WebSphere. For information about WebSphere profiles and port numbers, please refer to Chapter 4, “Installation and configuration” on page 63. You can use an existing profile or create a new one.

1. Make sure the WebSphere Application Server is started and access the WebSphere Administrative Console at the following URL:  
`http://localhost:9060/ibm/console`
2. Select **Resources** → **Mail Providers** → **Mail Providers** → **Built-in Mail Provider**. Click **Mail Sessions** in the Additional Properties section and create a new mail session with the following characteristics.

*Table 8-6 JavaMail session configuration*

Parameter	Value
Name	xPetstoreMail
JNDI name	mail/xpetstore/MailSession
Mail transport host	localhost
Mail transport protocol	SMTP
Mail store host	localhost

Parameter	Value
Mail store protocol	POP3

3. Click **OK** to create the mail session and save the configuration changes.

## Configuring the datasource in WebSphere

Since xPetstore requires a database, you need to provide that database configuration in the application server. Follow these steps to configure the datasource from the WebSphere Administrative Console.

1. Ensure that the WebSphere Application Server is up and running.
2. Click **Resources** → **JDBC Providers** → **New** and configure a new JDBC Provider with the following characteristics.

Table 8-7 JDBC provider values

Parameter	Value
Database type	DB2
Provider type	DB2 Legacy CLI-based Type 2 JDBC Driver
Implementation type	XA datasource

Click **Next**.

3. On the next page, specify the path to the JDBC driver and click **Apply**.
4. Click **Data sources** in the Additional properties section and create a new datasource with the following characteristics.

Table 8-8 Datasource values

Parameter	Value
Name	xPetstoreDS
JNDI name	jdbc/xPetstoreDS
Database name	to_xp_se

Click **Apply**.

5. Click **J2EE Connector Architecture (J2C) authentication data entries** in the Related Items section and configure a new J2C authentication data with the following characteristics.

Table 8-9 J2C alias configuration

Parameter	Value
Alias	DB2user
UserId	db2admin
Password	its0ral0

Click **OK**.

- Save the configuration by clicking the **Save** link shown in the message window at the top of the window. Then click **Save** in the confirmation window.
- Configure the datasource to use the component-managed authentication alias you just created. Click **Resources** → **JDBC Providers** and select the JDBC provider you created.
- Click **Data sources** in the Additional Properties section and select the **xPetstoreDS** datasource.
- In the **Component-managed authentication alias** pull-down menu, select **DB2user** and click **OK**.
- Save the configuration by clicking the **Save** link shown at the top of the message window. Then click **Save** in the confirmation window.
- Test the connection by selecting the **xPetstoreDS** datasource and clicking **Test connection**. You should receive a message saying a test connection for datasource xPetstoreDS on server1 was successful.

## Deploying the application

Now that you have the war file and have created all the resources necessary to deploy the application, log in to the WebSphere Administrative Console and go through the following steps to deploy the application.

- Click **Applications** → **Install New Application**.
- Select the war file to deploy, specifying the following values, and click **Next**.

Table 8-10 Installation values

Parameter	Value
Specify path	<source_home>\dist\xpetstore-servlet.war
Context Root	/xpetstore-servlet

- Then select **Generate default bindings** and **Use default virtual host name for Web modules**. Click **Next**.

4. You may see an *Application Security Warnings* window. Click **Continue**.
5. In the *Step 1: Select installation options* window, leave all the default options and click **Next**.
6. In the *Step 2: Map modules to servers* window, leave all the default options and click **Next**.
7. In the *Step 3: Map resource references to resources* window, select the **jdbc/xPetstoreDS** datasource from the drop-down menu in the `javax.sql.DataSource` section.
8. Select **DB2user** as the default authentication method from the *Select authentication data entry* drop-down menu.
9. Check the box next to the **xpetstore-servlet.war** module and click **Apply**, then click **Next**.
10. You may see an *Application Resource Warnings* window. Click **Continue**.
11. Select the default options in the *Step 4: Map virtual hosts for Web modules* window and click **Next**.
12. Click **Finish** in the *Step 5: Summary* window. It may take a minute to deploy the application and validate all mappings. Once finished, click the link **Save to Master Configuration** at the bottom of the window. Click **Save** in the confirmation window.
13. The application was deployed but is not started yet. Click **Applications** → **Enterprise Applications**, select the checkbox next to the **xpetstore-servlet\_war** application and click **Start**.
14. Check the installed application at the following URL:  
`http://localhost:9080/xpetstore-servlet`

## Testing the application

Since we do not have any kind of automated test, we have to test the application manually, following the same steps we performed to test the application running in Tomcat.

Once again, we signed in using `user1` and `password1` and browsed the different animals and races, and added one to the cart. We reviewed the cart items and checked out. This simple procedure take us through the application with no errors. We checked the log files and did not find any error or warning. The results were the same, so we can say that the application was successfully migrated.

## Development tips for portable applications

In this appendix, we provide a high-level overview, and a couple of more detailed examples, of some of the more common best practices that migration projects, or any other project, will benefit from following.

This appendix is organized in the following sections:

- ▶ End-to-end life cycle
- ▶ General development best practices
- ▶ Java development best practices
- ▶ Enterprise Java development best practices

## End-to-end life cycle

The WebSphere Application Server V6 environment and its tie-in to other Rational tools offers the developer support at every stage of the application development life cycle.

Key stages in this life cycle are:

- ▶ Requirements gathering and analysis
- ▶ Prototyping
- ▶ High-level design
- ▶ Low-level design
- ▶ Implementation/coding/debugging
- ▶ Unit testing
- ▶ Integration testing
- ▶ Functional verification testing
- ▶ Independent testing
- ▶ Acceptance testing
- ▶ Performance testing
- ▶ Deployment
- ▶ Maintenance (including fixes, modifications, extensions)

## Software development life cycle key concepts

This section looks at the fundamental concepts in the software development life cycle. Trends in the industry have moved from a static waterfall model to a more dynamic iterative model. Within these models, quality has to be assured by a process of verification and validation.

### Waterfall model

The life cycle can be looked at as a waterfall model because the output of each stage spills into the subsequent stage. For example, once the high-level design has been created, the artifacts created during that stage, for example the design of a user interface, feed into the low-level design of the modules needed to implement that user interface. This waterfall way of looking at the application development life cycle is largely non-iterative. This means that each stage is started when another stage finishes and there is little or no overlap.

### Iterative model

An iterative life cycle model addresses the same stages as the waterfall model but there is some degree of overlap. *Iterative* means that the cycle can feed back into itself and that software grows as the life cycle is repeated.



This iterative behavior occurs at a macro and micro level. At a macro level, the entire life cycle repeats itself; the maintenance stage often leads back to the requirements gathering and analysis stage. At a micro level, the review of one stage may lead back to the start of the stage again or indeed back to the start of another stage.

For example, if during low-level design, a flaw or inconsistency is found in the high-level design, then a micro HLD stage is followed again to ensure that the HLD is updated correctly. Indeed, this may even feed back to a micro cycle of the requirements gathering and analysis stage if the flaw in the HLD was found to be due to vagueness in the requirements analysis.

### **Verification and validation**

The micro iteration approach involves the concepts of verification and validation (V&V). Verification means to prove something is correct. Validation means to prove something is sound and logical. In software and systems development the two words map respectively to two key questions:

#### ***Are we building the right system?***

This is the verification question. Is the system that we are building correct in respect to its requirements? This question can be answered by referring to the requirements analysis and design documentation. If anything is not clear or is inconsistent from these stages, then these stages require a further iteration.

#### ***Are we building the system right?***

This is the validation question. Is the system soundly and logically constructed? Is it robust? Is it maintainable? Does it conform to standards? This question can be answered with varying degrees of formality. For example, the logic of an algorithm might be reviewed informally by a development team. At the other extreme, the logic of an algorithm might be formally proved using mathematical methods (such as the mathematical formal method language Z).

### **Quality assurance**

Quality assurance in software development is achieved by having some process of V&V in the development life cycle. This, in turn, involves some kind of cyclical review process where something is produced and then is either validated or verified. This process is iterated upon until a satisfactory level of quality is achieved.

## **The Rational Unified Process**

Rational software development tools are all built around the Rational Unified Process (RUP) to a lesser or greater extent. The Rational Unified Process is an iterative software development process. It is iterative at a macro and micro level.

At the macro level, phases of *Inception*, *Elaboration*, *Construction* and *Transition* can be identified in the process. These phases are basically periods of initial planning, more detailed planning, implementation and finalizing and moving on to the next project cycle. The next cycle will repeat these phases. At the micro level, each phase may go through several iterations of itself. For example, during a construction phase, coding, testing and re-coding may take place a number of times. Figure A-1 gives an overview of the Rational Unified Process.

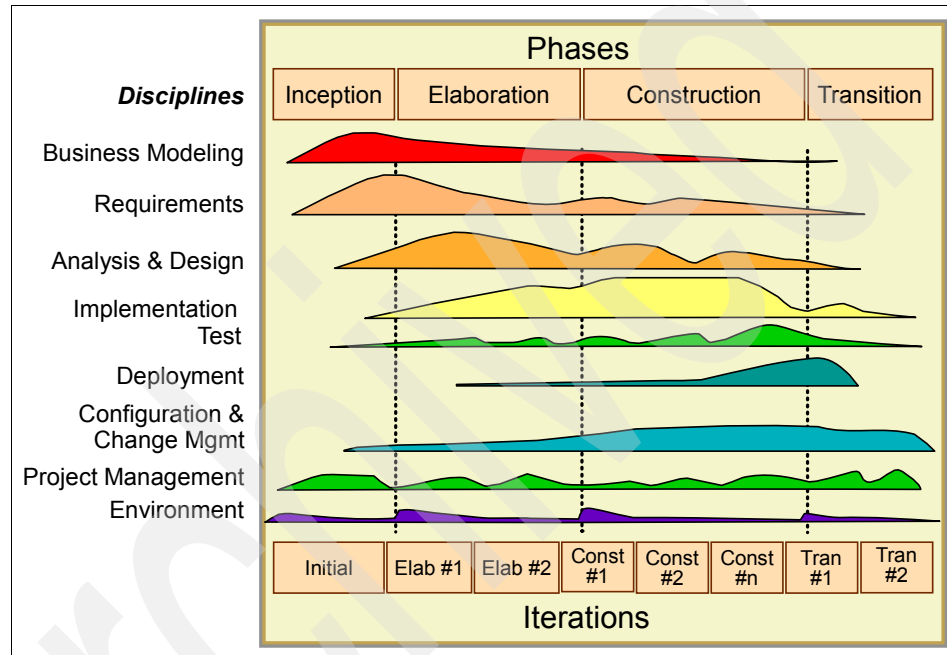


Figure A-1 Rational Unified Process overview

As also shown in Figure A-1, the Rational Unified Process identifies a number of disciplines that are practiced during the various phases. These disciplines are practiced during all phases but the amount of activity in each phase varies. Clearly, the requirements discipline will be more active during the earlier inception and elaboration phases, for example.

The Rational Unified Process maps disciplines to roles. There are many roles but the roles break down into four basic sets of roles: *Analysts*, *Developers*, *Testers*, *Managers*. Members of the team may take on more than one role. More than one team member may have the same role. Each role may require the practice of more than one discipline. Table A-1 on page 299 shows the basic mappings between roles and disciplines.

Table A-1 RUP disciplines (vertical) mapped to RUP role sets (horizontal)

	Analysts	Developers	Testers	Managers
Business Modeling	X			
Requirements	X			
Test	X	X	X	
Analysis & Design		X		
Project Management				X
Configuration & Change Management				X
Process Configuration & Project Environment				X
Deployment				X

**Note:** Each discipline and each role in Table A-1 has many sub-roles and sub-disciplines. The table is simply an overview.

The Rational Unified Process can be followed without using Rational Software; it is just a process specification. However, the Rational Unified Process provides specific guidance (called Tool Mentors) on how to use Rational Software when following the process. The disciplines identified in the Rational Unified Process such as requirements analysis, design or testing map to specific pieces of Rational software and artifacts that this software generates. The RUP is a process that can be 'bought into' as much or as little as is required.

For more information about the Rational Unified Process, visit the following URL:

<http://www.ibm.com/software/awdtools/rup>

# General development best practices

Any development effort would benefit from the following best practices. Emphasis is placed on an iterative and object oriented approach because these are the prevailing trends in contemporary project management and development.

## Iterative development

During development, there are always unknowns. These unknowns can take the form of the use of new technology, the use of new development staff or the use of new ideas to solve problems. These unknowns then take the form of risks. The earlier these risks are exposed, the earlier they can be mitigated.

Developing iteratively means to go through a number of sub-development cycles in turn in order to expose risks early. This is in contrast to the waterfall model. In a waterfall development model, because testing occurs late in the cycle, risks (such as the use of new technology and its unknowns) are not exposed until the components that use that technology are built. Worse, they may not be exposed until those components are unit, integration and acceptance tested.

Iterative development steps through sub-iterations of the entire development process. This means that fundamental parts of the application are prescribed, designed, implemented, tested and deployed in very controlled iterative cycles.

The key word here is *controlled*. The objective of the iteration has to be very clear and self-contained. Discipline in project management and software development throughout the micro iteration has to be as tight as it would be in the entire macro iteration.

For example, a project called RedbookApplication might require the creation of a Web front end using Struts technology, an EJB back end including the use of Java Messaging Service technology and an EJB client application. The front and back end layers may require many interface and back end components. However, a first iteration might simply design, code and test a single Web interface component, a single back end component and a single EJB client component. The Web interface would make use of Struts in this first iteration and the back end component would make use of Java Messaging Service technology at this early point.

This first iteration is not implementing all the Web interface, backed EJB and EJB clients needed. But what it is doing is creating a single example of each. What it is also doing is creating parts of the application that will be needed rather than prototyping with functionally irrelevant example code. Finally, and perhaps most importantly, it is implementing and testing the use of the specific technologies

such as Struts and Java Messaging Service which may be new to some or all members of the development team.

Furthermore, there may be novel ideas and or new design patterns that are being used. An example of these should be included also in this first iteration.

It is important here to note that the iteration is producing functionality that is needed in the final application. The iterative approach is similar in some ways to prototyping. Prototyping is the practice of building an application to prove theories from the design stage. It is also possible to show a prototype to end users and or customers in order to ask: “Is this the sort of thing you wanted?” It is commonly a stage somewhere between requirements gathering, design and coding stages in the waterfall model.

The danger of prototyping are that while prototyping produces an application that tests theoretical and new unknown areas, it is not focused on necessarily producing functionality required in the end product; it is often discarded, which in some ways can be seen as a waste of development effort. On the other hand, and in the worst case, if the prototype is concerned with producing final functionality, it may do so in a way that is not robust or well designed, just to get something up and running. This poor design and lack of robustness can then find its way into the final product. It has happened that clients, once shown a prototype, think it is so good (and cheap, because they have something that “works” in a short period of time) that they want to use that as the final product.

The iterative approach takes the benefits of prototyping but avoids its pitfalls. The deliverables of the iteration are deliverables for the final product. By following the iterative approach, risks are identified early and mitigated. There are other benefits in developing iteratively beyond risk mitigation:

### **Absorbing changing requirements**

The iterative approach also allows changing requirements to be absorbed. Requirements will change. Life is not static and even during a short development project, there is enough time for customers or end users to change their mind. These changes may have good reasons behind them. The environment in which the application is to be used may change. The market to which the business is responding may change. The experience of the end users or domain experts may change (often as a result of focusing on the requirements for the application). These changes are not necessarily to be discouraged as they can lead to genuine and necessary improvements in the original requirements and designs. If an iterative approach is being followed then these changing requirements can be absorbed more easily during the current iteration.

## Quality

Quality improves through iterative development. By definition, the application goes through a number design, coding and testing cycle revisions; this enhances the application's robustness, performance and maintainability.

## Learning

Members of the team learn lessons. They can apply the knowledge in the same macro development cycle or project. If a team member needs specific training, these needs are identified earlier on and the knowledge learned can be applied in the same project. Team members also learn about the development process itself. The process or the team member's practice of the process can be tweaked and improved upon.

## Reuse

Reusable components, or common components, may turn out to be not so generic in practice. Likewise, some components, which were thought not to be generic, may emerge as generic as other uses become apparent. These design considerations may not emerge until the project is well under way. Also, the use of third party technology (for example a GNU API such as Apache Xerces XML API) may be evaluated. Sometimes, third party technology can turn out to be inappropriate and another alternative needs to be sought; sometimes the technology may have additional, unexpected benefits that one would like to take advantage of too. The iterative approach makes these adjustments possible.

In summary, software development projects are learning processes no matter how well planned they are. The iterative model allows for controlled and sub-iterations of the entire life cycle with very clear goals for each sub-iteration. This iterative development model allows the team to take advantage of new knowledge earlier on. Hindsight is taken advantage of before the macro-iteration of the project life cycle ends.

## Requirements definition

Capturing requirements and managing changes to those requirements is fundamental to a successful project. Clear requirements and a process for managing changes to those requirements gives developers a chance to design and implement the right system. If requirements are vague and the process for managing changes to requirements is poor or non-existent, then at best the deliverables are going to be late and at worst they are going to be late and incorrect. The use of use cases is a best practice when modeling requirements. These use cases can directly be used to design components and devise test cases.

A use case is a scenario in the application that has actors and outcomes. It is a model of the permutations of events that may occur or be allowed to occur during that scenario. Use case diagrams are a part of the Unified Modeling Language (UML) and are a standard way of expressing these scenarios. It is possible to write use cases without creating use case diagrams. It is highly recommended that use cases be identified and created during the requirements stage to achieve a successful development project outcome.

## Object Oriented (OO) approach to design and programming

This seems an obvious statement, given the use of contemporary OO languages, but it is an important point to make because, as is commonly said, it is all too possible, but undesirable, to use OO languages in a procedural way (just as it is possible to write OO programs using procedural languages).

Developers using OO languages do not necessarily follow OO best practices nor think in a component-based manner. Developers must think in a modular way during design, coding and testing. The application is a module made up of smaller components working together. The key concepts that should be considered are *cohesion* and *coupling*. Developers should be aiming for high cohesion and low coupling in modules.

High cohesion means that areas of common data and behavior are encapsulated in a single object or related set of objects. This approach groups behavior and data into manageable and reusable components.

Low coupling means that the interfaces of each component that allow the components to interact should be kept as simple as possible and calls to interfaces should be kept as few as possible.

**Tip:** The success of the approaches is cumulative. If the right decisions are made in making modules and groups of modules cohesive, then the goal of low coupling is a good deal easier to achieve.

If you do not understand this concept then simply consider the placing of a fire hose on an ambulance rather than on a fire truck. No doubt the ambulance will sometimes be in the same place as the fire truck and the firefighters can use the hose. But clearly, it would be better if the hose were part of the fire truck. This may seem like common sense but in the abstract world of software development, such clarity can sometimes be harder to come by. When in doubt, applying first principles such as the laws of high cohesion and low coupling can help clarify the situation.

Good OO approaches and thinking lead to elegance, reusability and easier maintenance. Good OO approaches and thinking are surprisingly rare among developers using OO languages and technologies. Use of design patterns and IDE tools can help guide the developer into good OO practices but understanding OO design principles is fundamental.

**Tip:** There are two basic OO principles that we especially recommend because of the many benefits that are associated with their use:

- ▶ Coding to interfaces

This principle is coding to interfaces instead of concrete implementations. The interface is the contract between the provider of a service and the client. Changing the implementation does not break this contract; only a change in the interface can do this.

- ▶ Object composition

This principle can be seen as an extension to the first one. Using inheritance instead of object composition causes problems, because subclasses directly depend on the parent class. A change in the parent class might break classes down further in the inheritance hierarchy.

A better strategy is to use object composition in combination with the principle of coding to interfaces. This results in finer-grained objects that are used through a façade that hides the implementation details. The objects are assembled at runtime, by Dependency Injection or code, which has the added benefit of allowing you to easily switch the implementation.

## Modeling languages

Unified Modeling Language (UML) is a visual way of expressing concepts in software development. Use cases and class architectures, for example, can be expressed unambiguously. Unified Modeling Language does not have to be used but developers need to have a way of communicating design ideas clearly and without extraneous details. Unified Modeling Language class diagrams, for example, are one view of the OO component architecture. They show the entities that the developer wants to show and do not show those that would confuse issues. Rational Rose and Rational Application Developer V6 both have UML diagram capabilities. It is recommended that developers use this functionality.



**Note:** UML can be used passively, especially with class diagrams. This is important because developers new to UML will find it far easier to read diagrams than to create diagrams. Components such as classes and packages can be created in Rational Application Developer V6 (and other Rational Tools such as Rational Rose). These components can then be dropped onto a diagram panes and the class diagram UML is automatically generated. These diagrams can then be exported to design documents. The auto-generation of UML means that even developers that are relatively new to UML can use it. Developers can passively read the diagrams rather than actively produce the diagrams.

## Regular reviews and check points

During development, design and code reviews are invaluable. Without them, quality suffers. Designs must be reviewed against requirements. Code must be reviewed against designs. Code must also be reviewed for other quality standards. We will look at some of these standards later in the section on Java Development best practice. Design and code reviews can have varying levels of formality. What is key is that the review process should:

- ▶ Schedule time for developers to collectively look at specific design or code modules.
- ▶ Record findings formally so that there is a permanent record of what needs to be changed.
- ▶ Schedule time for developers to make the changes to the design and code artifacts.

Reviews should be held regularly as per the iterative approach. This keeps developers 'on the same page' and corrects problems early on. Regular reviews also keep reviews fresh by reviewing smaller chunks of code. Reviewing all the code at the end of implementation can be so monotonous, given the amount of code to review, that the quality of the inspection is significantly reduced.

## Java development best practices

Java will naturally be the development language of choice in an Enterprise Java Environment. Java Standard Edition is the core platform for all Java development projects including Enterprise Java projects. Here we look at best practices that apply to all projects using Java Standard Edition. Some of these practices are also good practice for any development project.

## Code documentation

Undocumented code is next to useless. Documentation best practice is fundamental to the design, implementation and maintenance phases. If code is not properly documented then code maintenance in subsequent iterations can be at best problematic, and at worst, impractical. Code maintenance has many sub areas. Extending applications, scaling applications and fixing applications are all forms of maintenance. Without good documentation, these tasks are much harder to complete.

### JavaDoc gives developers a contract

JavaDoc is used to document classes, instance variables and methods. It has very clearly defined annotations that can aid the development process considerably. For example, the method JavaDoc comments have annotations to describe input parameter values, output values such as return values and exceptions and under what circumstances these are produced. As such the method JavaDoc comment, if properly implemented, represents the method contract.

Given that JavaDoc gives such a precise way of documenting the modules in an application, JavaDoc documentation of code is not something that should be left to chance. Indeed, JavaDoc comments are artifacts in the development process that should be written at design time. There are various ways of doing this.

A low-level design document may simply describe all the modules that are to implement the system. The JavaDoc comments can then be written into the low-level design document. When it comes to coding. The low-level design then forms the template for developers who have been allocated particular modules. They can cut and paste the JavaDoc comments into their source files.

A design tool such as Rational Rose can be used to design components. As components are created, they can have their JavaDoc class, instance variable and method comments filled in. When the design is complete design documentation can be generated in the form of UML diagrams via automated design documentation tools such as Rational SoDA®. Additionally, Java class shells can be generated. That is, Java source code can be generated from the Rational Rose® model including all the class, instance variables and method signatures along with all the JavaDoc comments. These classes can then be allocated to each developer for implementation.

Whether using design documents with JavaDoc comments in them or more sophisticated design tools such as Rational Rose that generate code shells, the outcome of this approach is that the JavaDoc comments which form contracts are placed in front of the developer before they start coding. This keeps the contract for the module in front of the developer as he codes the module. By

sticking to the class and method contracts the developer is more likely to achieve the desired outcome: a module that is going to pass Unit Test first time around.

### **Local comments best practice: why, not what**

During coding, code should be commented where necessary. The comments we are talking about in this case are not JavaDoc comments. They are local comments written at some point in an algorithm inside a method body. The general best practice rule of thumb for writing effective comments is that the comments should explain 'Why?' and not 'What?'. Well-designed, well-written code is itself a documentation of what is happening. If code is so poorly designed or esoterically written that it is not clear what is happening then the developer should consider being more explicit in their coding style or re-factoring the code. Why something is being done is not always so clear locally. Variables and method calls often come from separate modules. Non-procedural languages do not put everything in front of the developer at the same time. Explaining why something is being done in a particular way is what local comments should be used for.

### **Saving time in code reviews**

Good documentation of code saves time during code reviews. It allows other developers to read code more easily. It also cuts down on findings from reviews. One of the major, and unnecessary, types of finding in code review is the need for more accurate JavaDoc and local comment blocks, if those blocks are found at all.

## **Unit testing**

Unit testing is the testing of discrete modules to ensure that their interfaces take the correct inputs and give the correct outputs on the basis of those inputs.

Having properly implemented unit tests results in code has many benefits compared to untested code including:

- ▶ Easier to detect a broken build  
Run the tests automatically at each build. If the tests are not completed successfully, the code cannot be deployed.
- ▶ The application has at least two clients  
The test case and the application itself. This leads to better designed code which is less tightly coupled.

- Easier to refactor

It is easier to make the decision to refactor the code, since you have test cases that automatically verify that your application is working after the refactoring is done.

- The application is better documented

By creating unit tests that use your code, you are automatically documenting how the code is used.

- Easier to verify migration

It is easier to verify that the application works after having migrated it.

Next, we introduce two tools that can be used for automating tests, and one which helps in assessing the amount of code that is being tested.

## JUnit

Rational Application Developer V6 includes support for implementing and running unit tests implemented with JUnit. There are wizards for generating test projects that use test templates for testing different types of modules. Templates for simple method testing, interface testing and EJB testing are all supported. In each case, JUnit test cases are generated and can be tweaked by the developer once generated.

JUnit is also available as a standalone package that can be downloaded and run outside of Rational Application Developer V6, for example Ant build scripts.

For further information about JUnit, refer to the Rational Application Developer V6 product documentation, visit the following URL:

<http://www.junit.org>

The following sections discuss some JUnit best practices.

### ***Code unit tests first***

Planning and implementing unit tests before the actual application code is sometimes referred to as Test Driven Development (TDD). TDD is an agile process that provides many benefits including helping you implement code that is properly tested and that is less tightly coupled.

Tests that are written according to the TDD process usually follow these simple steps:

- Plan the test case

What do you want to test and how? Plan the assertions that are needed to verify that the test is successful.

- ▶ Write a test that satisfies the goals of the plan but fails  
Write the JUnit test method, for example `testIncreaseSalary`, and the assertions needed to verify the test:  

```
assertTrue( person.getSalary() == (previousSalary + increase) )
```
- ▶ Write the least amount of code required for the test case to run successfully  
Write code and business logic for the test to pass.
- ▶ Refactor the code  
Remove any code duplication and code smell.
- ▶ Continue writing the next test case  
Implement and test the application in small increments. Use a code coverage analysis tool like Clover. Clover can be run against the code base and will tell you the amount of code covered by unit tests.

### ***Make JUnit tests simple***

JUnit tests should be simple in that they should test a discrete area of functionality. This means that areas of functionality can be tested in isolation. If larger areas of functionality need to be tested then test cases can be written to run groups of test cases.

It is also a good idea to keep the JUnit test cases so simple that they are executed quickly.

### ***Make JUnit tests generic***

The whole point of having a set of unit test modules is that they can be run at any time to validate the system and all of its sub components. This will not be the case if the tests are not generic. By generic we mean that the test is:

- ▶ Not dependent on any particular external conditions or state.
- ▶ Not dependent on OS features such as file system idiosyncrasies.
- ▶ Not dependant on outcomes or data from other tests.
- ▶ Not dependent on local time zones.

## **HttpUnit**

HttpUnit is a framework that provides tools for automating Web application testing. Using a tool like HttpUnit, in combination with JUnit, allows you to perform black-box testing of J2EE applications, by accessing the application in the same way the end user would, in contrast to JUnit which can only test code through the use of Java APIs.

HttpUnit is implemented in a similar way to a simple browser and allows you to simulate user interactions through the use of the HttpUnit API.

For further information about HttpUnit, refer to the following URL:

<http://httpunit.sourceforge.net>

## **Clover**

Clover is a code coverage tool that can be used for producing reports that show the percentage of packages and classes that are covered by unit tests. This tool allows you to identify code that is not being tested at all or that is not sufficiently tested. It is a good idea to keep the unit test coverage above fifty percent, but remember that the more unit tests you have the more complexity you have to deal with. Having a code coverage of 100% is not a sensible goal.

For further information about Clover, refer to the following URL:

<http://www.cenqua.com/clover/>

## **Automation**

Processes involving manual tasks are often prone to errors and are not easily repeatable due to the human factor involved. This is why it is a good idea to automate as many software development processes as possible.

Processes that can, and in most cases should, be automated include:

- ▶ Build process
- ▶ Unit and acceptance tests
- ▶ Administrative tasks

### **Build process**

The build process is perhaps the most error prone and complex task that can be automated. Usually the build process involves tasks such as compiling source code, copying and modifying configuration files and creating a deployable package.

Automating the build will give you the benefits of having a repeatable build process, since you are not depending on a person clicking the right button at the right time. It will also allow anyone to

Two popular frameworks support the goal of build automation. Both are top-level Apache Foundation projects:

- ▶ Ant

Apache Ant is a Java based implementation similar to Make. The main difference is that Ant uses XML for build scripts and is easily extensible.

- Maven

Apache Maven works on a higher level of abstraction compared to Ant. Removing the need for writing common Ant scripts, although Ant scripts can still be used within Maven. Another difference, when comparing with Ant, is that Apache Maven is not just a build tool but a project management tool that can be used for documentation and reporting purposes.

Either one of the tools can be used not just for build automation but for the automation of almost any process involving scripting.

### ***Continuous integration***

Continuous integration tools help you in automating the manual tasks involved in the process of making daily builds. Making daily builds has the benefits of integrating changes to source code early and often. This allows you to detect potential problems with the code base more easily and earlier, which is useful in big teams where many people are working on the same code base at the same time. Integrating code changes less frequently can result in integration problems.

The process of making a daily build involves manual tasks that are usually executed by a build manager. The process typically includes the following tasks:

- Running the automated build

Process of building a deployable package automatically, with the help of a build automation tool.

- Testing the build

Running automated tests, based on for example JUnit and HttpUnit. Detecting a broken code base and reporting it to the team.

- Version control process

Includes tasks like integrating changes to source code, labeling and branching.

- Generating reports

Creating and distributing reports containing information about the current state of the latest build and change log to all team members.

There are several open source tools that support the continuous integration concept. Most of these supports build tools like Ant and Maven, different version control systems and reporting options. The following is a list of some of the more popular tools:

- Lunt build

<http://www.pmease.com/luntbuild/>

- ▶ CruiseControl  
<http://cruisecontrol.sourceforge.net/>
- ▶ DamageControl  
<http://damagecontrol.codehaus.org>

## Enterprise Java development best practices

This section looks at best practices specifically in the Enterprise Java field and also with some WebSphere Application Server V6 specific concerns.

### Modularization

You should consider modularizing your application, whichever technology you choose to use to implement your applications. In J2EE this is enforced by the specification itself, which assigns an archive type for each different kind of module in the application. Inside of an EAR file, you are basically allowed to have as many different modules as you need. Try to separate different types of concerns into modules that are completely independent of each other.

When you design your modules, you should keep in mind that every module should be independent, and that the module will be placed in a dependency hierarchy. For example, an EJB module must not depend on a WAR module, which is not always the case in real life projects. UML diagrams will help in visualizing the whole application and creating independent modules.

The main benefits of using modules are:

- ▶ Encapsulation  
All the data and necessary functions are contained in the same module.
- ▶ Separation of concerns  
Every module contains classes related to only one kind of problem or solution. This will ease the problem determination.
- ▶ Traceability  
Every module will contain a set of classes related to a set of use cases. This relation will make the evaluation of changes in a use case easier.
- ▶ Testability  
You can build automatic tests for each module more easily, since you only need to use one module.



- Work division

It is easier to assign a module to a developer or group of developers than the whole application. Furthermore, it is easier to assign one module per developer than a big module to a group of developers.

- Migration

As discussed in 3.1.3, “Strategies to handle migration complexity” on page 33, it is easier to migrate the application module by module, than migrate it all at once.

## Follow the standards

This is most important practice when we talk about migration. The use of vendor-specific code will be translated into modifications, re-engineering and redesign of source code in some cases. In most cases, the proprietary code is a way of simplifying a task, but surely the time you gained in that task will be spent twice during a migration, not only from vendor to vendor, but from version to version, and that kind of migration will be necessary some day.

Even when you are not using vendor-specific code, some practices could be problematic at migration time, for example transaction management and program-to-program communication. The advice here is not to reinvent the wheel. When you need to solve a problem, look for solutions covered by the J2EE API instead of implementing your own.

There are some limitations in the specification that could make you deviate from this advice. Be sure to have a sound reason for not using the specification and that there are no alternative solutions to your problem within the specification.

**Note:** In most cases, it is also possible to use open source frameworks instead of the J2EE standard APIs and services provided by the application server. One example would be using Hibernate for database access instead of entity EJBs.

It is difficult, if not impossible, to be completely sure that the framework will work on another application server without problems. This is mostly because of small differences in how the different application servers vendors implement the J2EE specification.

If it works, you are likely to have less migration issues than when using the J2EE APIs, but if not, you will have to fix the issue yourself or file a bug report and wait for it to be fixed.

## Design patterns

If you have a problem to solve then you can depend that it has been faced and solved before. Commonly used solutions in application architectures fall into categories. These categories are called *design patterns*. The following are generic design patterns that are not specific to J2EE nor to the Java language. But they are design patterns that are very extensively used in J2EE. For further information about J2EE design patterns, visit the following URL:

<http://www.theserverside.com/patterns>

**Note:** It is important to know when to use a pattern and how to apply it correctly. This knowledge can only be gained through experience and practice. Java and J2EE applications tend to overuse patterns, which leads to unnecessary complexity.

### Model, View, Controller (MVC)

The Model, View, Controller pattern originates from tools developed for the Small Talk programming language. The MVC pattern follows good OO design principles. That is, it promotes high cohesion and low coupling.

Most applications have interfaces and business logic. Interfaces can be human user interfaces such as command lines and GUIs. Interfaces can also be middleware interfaces (which are essential application to application interfaces).

Business logic and back ends deal with resources interaction such as interaction with databases.

In order to make applications highly maintainable (which means that they are extensible and easy to understand and modify regardless of future requirements), it is desirable to keep the coupling of interfaces and back ends as low as possible. The MVC pattern prescribes that the interface and the back end are entirely separate and that all communication is done via a controller layer. This controller layer introduces indirection into the application. This means that any type of interface (a GUI, a command line or an application interface) can be coded to talk to the controller layer. In addition, any back end application layer can talk to the controller layer too.

MVC is fundamental to Enterprise Java. In the simplest form, JSPs and EJB client applications represent the View. Servlet and Session Façade layers represent the controller and entity beans represent the Model.

However, as with OO approaches and standard Java, just because a developer is given the tools to create OO or MVC architectures does not mean that the developer will create OO or MVC architectures. It is possible, for example, to use a JSP to call an Entity EJB directly. This is logically correct but may not be the

best design solution in terms of keeping the interfaces between the back and front ends simple. During development, developers should keep the MVC goal in mind and be aware that it is still possible to break this pattern despite of using J2EE technologies.

### **Struts**

Struts is a framework that is implemented according to the MVC pattern. Prior to Struts, developers used combinations of JSPs and Servlets to create the View and Controller layers of an application. Struts took the best practices that emerged from the JSP and Servlet architectures and created an architecture that introduces an action model where JSP form data is submitted to an ActionServlet controller.

Struts uses JSP tag libraries extensively. WebSphere Application Server V6 now comes with caching features specific to Struts architectures to reduce the overhead of tag library references.

Where possible, JSP rather than XSLT should be used to render the view layer as XSLT (which is the transformation of XML to HTML pages) is expensive.

Form Beans should not be passed to the business logic layer. The layers should be completely separate. Passing form beans to the back end would mean that the back end is not independent from the front end in terms of the objects it needs to know about.

**Note:** Struts is the de facto standard for MVC frameworks, but there are many other open source frameworks that are more light-weight and easier to use. Java Server Faces (JSF) is a standardization effort that tries to unify a fragmented market.

### **Java Server Faces (JSF)**

Java Server Faces is a specification, defined by the JSR-127 Java Community Process, which aims at simplifying the creation of complex GUIs in Web applications. JSF is similar to Swing and other client APIs in that it provides a set of standard widgets and a framework for creating additional widgets.

Java Server Faces splits the user interfaces presentation and processing cleanly, which allows for easier construction of multiple user interfaces. The components available are varied and sophisticated which increases the possibility of what GUI components can be used in the front end. This richness of features and further level of indirection between the interface and processing comes at a cost. JSF can be relatively poor in performance when compared with JSP and Struts. The decision to use the technology has to weigh the merits of a sophisticated and extensible interface against the performance costs.

Consideration of how much data a JSF component may display and trying to keep that to a minimum can ameliorate the performance cost.

### **Data Access Objects (DAO)**

Applications that store and retrieve data from a persistent storage, like for example a database, will benefit from using the Data Access Objects (DAO) pattern. The DAO pattern is similar to a façade object which provides a simple interface to client modules in that it implements the connection and retrieval code internally.

The DAO pattern hides the implementation details from the client by encapsulating details related to data access. This makes it easier to change the type of persistent storage, to for example XML files from an RDBMS, and the data access mechanism, for example switching from entity EJBs to Hibernate. Using EJBs for accessing databases and other resources is not always necessary, and if this is the case, the memory and network overheads of using an EJB should be avoided.

### **Service Locator**

J2EE application servers expose services and components like, for example, EJBs through JNDI. A single object can be used to do JNDI lookups for resources such as EJBs and Java Messaging resources. Delegating this task to one object that is then called by other objects is far better practice than all objects doing the same JNDI call to look up objects.

### ***EJBHomeFactory***

This is a composite design pattern that uses other design patterns. Its purpose is to reduce the overheads when looking up home interface references for EJBs. This design pattern is similar to the Service Locator pattern. It also uses the singleton pattern, which means it maintains a single instance of itself for use by other modules. It also uses the factory design pattern (that is, its sole purpose in life is to produce instances of a particular object type for other modules).

### **Façades**

The façade design pattern offers a more simplified interface by encapsulating a more complex interface or collection of interfaces. A Session Façade pattern is constructed from a stateless Session EJB accessing, for example, one or more Entity EJBs. The Session Façade is based on the Façade design pattern. A Façade simplifies the interface of a more complex underlying system. In a Session Façade, the complexity of finding and calling a number of Entity beans is simplified to one call to a method on a Session Bean. This pattern, in turn, reduces complexity in client layers.

This pattern also reduces network overhead, caused by remote method invocations, by encapsulating multiple remote calls into one.

This pattern also reduces transaction handling overheads by not allowing view objects to access entity beans directly. When view objects access entity beans directly, separate transactions are created which is an extra overhead.

## Singletons

Singletons are classes that return an instance of themselves to calling classes. They only return a new instance of themselves to calling classes, if the single instance of themselves that they maintain has not been created yet. The advantage of this is the avoidance of static method calls. Static method calls are not a very clean OO approach. Moreover, ensuring that only one instance of a Class exists at any one time means all users of the object reference that object rather than maintaining their own instance. Separate instances represent a memory overhead.

However, a word of caution about singletons: singletons are only desirable if the object is totally generic in all situations for all calling classes, if not then the singleton pattern should not be selected.

## Factories

The factory design pattern is where an object has a dedicated purpose. The purpose is to produce instances of a particular object class. The Factory object is the single point of access for obtaining an instance of the object. The Factory design pattern is often used in conjunction with the Singleton pattern that ensures that only one instance of the Factory object exists in memory.

## Dependency Injection (DI)

Dependency Injection, also known as Inversion of Control, is a pattern that delegates the process of instantiating objects and setting up the dependencies between these objects to a light-weight container. The Dependency Injection pattern provides the following benefits:

- Removes hard coded dependencies between classes

The container takes care of the dependencies and the creation of objects.

Dependency Injection promotes the principle of coding to interfaces instead of concrete classes. Using interfaces allows you to change the implementation with just a configuration change.

- Configuration flexibility

Configuration of components is done in one central place, the container configuration. This makes it easy to change application details without recompiling source code.

- Improves testability

JUnit test cases usually do not depend on external systems like databases or a J2EE container. By coding to interfaces, you automatically improve testability. Changing the concrete implementation of an interface, for example changing a DAO to use an in-memory database instead of an RDBMS, is done in one place only, the container configuration.

There are several implementations of the Dependency Injection pattern that are available for download, such as:

- PicoContainer

PicoContainer is a light-weight container that implements the Dependency Injection pattern.

<http://www.picocontainer.org>

- Spring Framework

Spring Framework contains an implementation of the Dependency Injection pattern. Spring Framework is a more general J2EE framework, and includes many other features than just Dependency Injection.

<http://www.springframework.org>

**Tip:** Dependency Injection removes the need for implementing many patterns, including the factory, singleton and service locator patterns. Dependency Injection is a better solution than these patterns combined. By allowing you to create code that is less tightly coupled.

## J2EE best practices

The MVC pattern is the overriding design pattern in the J2EE application architecture. MVC is made up of cohesive Model, View and Controller layers of areas of functionality that communicate between each other through loosely coupled interfaces. Performance can be highly affected by the implementation of these layers and the ways that these layers communicate between each other.

In this section, we look at the best practices when designing and implementing these layers for high performance in a J2EE architecture. We also look at some more general issues related to performance.

### View layer: JSPs

Java Server Pages are HTML presentation layer pages with special embedded tags that are processed on the server side within special servlets. The servlets are automatically generated during deployment. An increasingly sophisticated library of standard tags is available which is, over time, reducing the amount of

Java code scriptlets that have to be used. The use of pure JSPs (rather than using Struts or Java Server Faces) gives the best performance. But Struts and JSF architectures have other benefits which should be considered.

Struts (which is an Apache org sub project) shields the developer from MVC architecture considerations and allows for better maintainability. JSF gives the developer a wide variety of specialized GUI components to choose from. Struts and JSF are now more comprehensively supported in WebSphere and Rational Application Developer V6 and Rational Web Developer V6 than ever before.

### ***JSPs and Servlets***

JSP and Servlet performance related best practices include:

- ▶ Minimizing the include JSP tag statements as each is a separate servlet.
- ▶ Use the *usebean* JSP tag only to access existing beans, not to instantiate new instances.
- ▶ If a session object is not required in the JSP, use the `<%page "session=false" %>` directive to avoid the default creation of a session object.
- ▶ Avoid the use of the `SingleThreadModel` for servlets or client requests will be sequential rather than running in parallel. The `SingleThreadModel` has been deprecated in version 2.4 of the servlet specification.
- ▶ Place expensive one time initialization calls in the place servlet **init** method rather than the `doGet` and `doPost` method calls.

### **Controller layer: servlets and session beans**

Servlets can be used to implement the controller layer which communicates with the presentation layer. The use of Struts and JSF generates this layer as part of those architectures which is one of the virtues of using those technologies. The following are best practices when implementing a controller layer:

- ▶ Keep presentation layer and business logic layer functionality out of the controller layer. For example the controller layer should not generate HTML nor should it process business logic algorithms.
- ▶ Place common functionality in the controller layer into a common super class.
- ▶ Each controller component should implement a specific, discrete task. For example, a separate servlet should be create for logging, in logging out, uploading data etc. Putting all controller tasks into a single servlet can lead to maintenance and configuration problems because the law of high cohesion has not been followed. That is, each area of functionality should be split into a discrete module: in this case, separated servlets.

### ***Maintaining state***

In general, HTTP session is used to maintain state specific to Web clients. Best practices for HTTP session objects are as follows.

- ▶ Minimize the size of sessions objects by storing only essential data for the application there.
- ▶ Invalidate unused HTTP sessions explicitly rather than relying on session time outs.
- ▶ HTTP Session should not be cached by the application or they will not be reclaimed when the session expires or is invalidated, this can lead to memory leaks.

Storing excessive amounts of data in the HTTP session will lead to performance issues. Depending on the server configuration, HTTP sessions can be persisted to a data store or replicated between cluster nodes, which involve serialization and network overhead.

Stateful Session Beans can be used to maintain state information too. Especially when the front end is not Web based. Such beans are not shared by multiple clients nor are they pooled. Stateful Session Beans exist until they are explicitly removed or until their session expires. Therefore, remove Stateful Session Beans when the client is finished using them, otherwise they will continue to exist in memory and or be passivated to disk which represent unnecessary overheads if the bean is no longer required. The bean can be explicitly removed by invoking its **remove** method.

### **Model layer: entity EJBs**

The model layer in an Enterprise Java application is where the real work is done. It is here that business rules and processing are implemented and where data is brought in and out of persistent storage.

### ***When not to use an EJB***

EJBs are objects that can be distributed, support transactions, can be pooled and support the J2EE security model for containers. If an object does not need to make use of these features, then the object should not be an EJB. EJBs are expensive resources in terms of communications and memory usage and should be used only when their advantages can be exploited by the application, not as a matter of course.



**Note:** When migrating the sample applications covered in this book, we had the biggest problems related with CMPs and EJBs in general. Mostly because of the complexity in using EJBs, vendor-specific features and database mapping problems. Another problem is that, in our experience, EJBs tend to be overused. This leads to unnecessary complexity and migration problems.

### ***EJB interfaces***

EJBs can be referenced remotely or can be referenced locally. Evaluate early on if a remote, distributed call is really required. If not, then use local interfaces; this makes an application far more efficient since the call is done by reference rather than by remote method invocation. This has been true since the EJB 2.0 specification. Before then, remote calls were used for all communications. Clearly the use of local interfaces will limit the deployment of the application to a cluster.

### ***Encapsulate data passed between distributed objects***

Distributed method calls are expensive. When making a distributed method call it is desirable to get as much data to and from the method as possible. Returning an object containing a collection of data, rather than a single data item is more efficient. If both options are required under different circumstances then both methods can be specified, but the use of the collection access method should be favored. This is far less network intensive than making a number of method calls for individual items. A single method parameter object that encapsulates a number of items that would otherwise be sent as separate parameters is also more efficient.

### ***Use Container Managed Transactions***

Use the container managed transactions functionality rather than developing your own transaction management. The container managed transactions execution is almost always more efficient and far easier to maintain.

### ***Avoiding memory leaks***

It is wrong to assume that Java manages memory and that therefore you will not get memory leaks in Java. Resource connections should be cleaned up carefully. Some tips for avoiding memory leaks are as follows:

- ▶ Use *finally* blocks to clean up resources, like database connections, properly.
- ▶ When using a singleton pattern set references to resources to null explicitly or they may not be released for garbage collection.

For more information about development best practices for performance, refer to *WebSphere Application Server V6 - Performance, Scalability, and High Availability*, SG24-6392.



## Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

### Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246690>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6690.

## Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
<b>trade3-WebLogic.zip</b>	Zipped Trade 3.1 sample source code and binaries.
<b>migrator_1.2.0.zip</b>	Zipped migration plugin for Rational Application Developer V6.
<b>ivata_db2_data.zip</b>	Zipped SQL with DB2 formatted data for loading the ivata groupware database.
<b>xpetstore-cmp-jboss-WRD.zip</b>	Zipped property files, XML, SQL and DDL files to ease the configuration and migration of xPetstore using WRD.
<b>xpetstore-cmp-wls.data.zip</b>	Zipped SQL with DB2 formatted data for loading the xPetstore EJB database when deploying and migrating to BEA WebLogic Server 8.1.

## System requirements for downloading the Web material

The following system configuration is recommended:

<b>Hard disk space:</b>	37 GB
<b>Operating System:</b>	Windows 2000
<b>Processor:</b>	Pentium IV 3.06 GHz
<b>Memory:</b>	2 GB

## How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 328. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Migrating WebLogic Applications to WebSphere Advanced Edition*, SG24-5956-00
- ▶ *Migrating WebLogic Applications to WebSphere Advanced Edition V4*, SG24-6179-00
- ▶ *Migrating WebLogic Applications to WebSphere V5*, REDP-0448-00
- ▶ *MySQL to DB2 UDB Conversion Guide*, SG24-7093-00
- ▶ *WebSphere Product Family Overview and Architecture*, SG24-6963-02
- ▶ *WebSphere Application Server V6 Planning and Design*, SG24-6446
- ▶ *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392-00
- ▶ *WebSphere Application Server V6 Migration Guide*, SG24-6369-00
- ▶ *Rational Application Developer V6 Programming Guide*, SG24-6449

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ WebSphere Extended Deployment:  
<http://www.ibm.com/software/webservers/appserv/extend>
- ▶ WebSphere Application Server - Express, system requirements:  
<http://www.ibm.com/software/webservers/appserv/express/requirements>

- ▶ WebSphere Application Server, system requirements:  
<http://www.ibm.com/software/webservers/appserv/was/requirements>
- ▶ WebSphere Application Server - Network Deployment, system requirements:  
<http://www.ibm.com/software/webservers/appserv/was/network/requirements>
- ▶ JUnit  
<http://www.junit.org>
- ▶ eclipse TPTP (Hyades)  
<http://www.eclipse.org/hyades>
- ▶ Enterprise Generation Language (EGL):  
<http://www.ibm.com/developerworks/websphere/zones/studio/egldocs.html>
- ▶ WebSphere Application Server Migration Resources:  
<http://www.ibm.com/developerworks/websphere/zones/was/migration.html>
- ▶ IBM Patterns for e-business:  
<http://www.ibm.com/developerworks/patterns>
- ▶ Using WebLogic RMI with T3 Protocol:  
[http://e-docs.bea.com/wls/docs90/rmi/rmi\\_t3.html](http://e-docs.bea.com/wls/docs90/rmi/rmi_t3.html)
- ▶ WebSphere MQ  
<http://www.ibm.com/software/integration/wmq>
- ▶ WebSphere - WebLogic Secure Interoperability:  
<http://submit.boulder.ibm.com/dd/wsdd/asis/501650/501650.html>
- ▶ Interoperating transactionally between application servers, WebSphere Application Server V6 Information Center:  
[http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjta\\_intop.html](http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tjta_intop.html)
- ▶ Migrating a WebLogic EJB Application to JBoss:  
<http://www.onjava.com/pub/a/onjava/2005/03/09/ejb-migration.html>
- ▶ XDoclet overview:  
<http://xdoclet.sourceforge.net/xdoclet>
- ▶ XDoclet tags for WebSphere:  
<http://xdoclet.sourceforge.net/xdoclet/tags/ibm-tags.html>

- ▶ WebSphere-specific deployment descriptors when using XDoclets:  
<http://xdoclet.sourceforge.net/xdoclet/ant/xdoclet/modules/ibm/websphere/web/WebSphereWebXmlSubTask.html>
- ▶ WebSphere Application Server Version 6.0 Information Center:  
<http://publib.boulder.ibm.com/infocenter/ws60help>
- ▶ IBM DB2 Universal Database for Linux, UNIX, and Windows Information Center:  
<http://publib.boulder.ibm.com/infocenter/db2help>
- ▶ IBM DB2 Migration Toolkit:  
<http://www.ibm.com/software/data/db2/migration/mtk>
- ▶ Rational Application Developer for WebSphere Software V6.0 Trial:  
<http://www.ibm.com/developerworks/downloads/r/rad>
- ▶ IBM Software Support:  
<http://www.ibm.com/software/support>
- ▶ WebSphere Application Server V6.0 Trial:  
<http://www.ibm.com/developerworks/downloads/ws/was>
- ▶ WebSphere Application Server support:  
<http://www.ibm.com/software/webservers/appserv/was/support>
- ▶ Profile creation wizard, WebSphere Application Server V6 Information Center:  
[http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tins\\_instances.html](http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tins_instances.html)
- ▶ DB2 Universal Database Express Edition V8.2 Trial:  
<http://www.ibm.com/developerworks/downloads/im/udbexp>
- ▶ Compatibility & Java Verification, Java AVK for the Enterprise:  
[http://java.sun.com/j2ee/verified/avk\\_enterprise.html](http://java.sun.com/j2ee/verified/avk_enterprise.html)
- ▶ BEA Product Documentation:  
<http://e-docs.bea.com>
- ▶ BEA Downloads:  
<http://commerce.bea.com>
- ▶ xPetstore download:  
<http://xpetstore.sourceforge.net>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

- ▶ IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

- ▶ IBM Global Services

[ibm.com/services](http://ibm.com/services)



# Index

## Symbols

@ejb 54  
@web 54

## Numerics

2-phase commit transactions 115  
3rd party JMS 37

## A

Activation Specification 106  
adaptable applications 2  
AdminApp 83  
AdminConfig 83  
AdminControl 83  
Administrative Console 78  
AdminTask 83  
Agent Controller 28  
AIX 12, 19  
Analysts 298  
Annotation-based programming 23, 27  
Ant 261  
Ant 1.5.1 5, 113  
Ant build script 145  
Apache Ant 261  
Apache Ant 1.5.1 5  
Apache Commons 4, 271  
Apache Log4j 100  
Apache Maven 261  
Apache Maven 1.0.2 4  
Apache Server 20  
Apache Software Foundation 258  
Apache Struts 1.2.4 4  
Apache Tomcat 5.5.9 90  
Apache Torque 267–268  
Apache Velocity 1.3.1 5  
Apache Xalan 93  
Apache Xerces 93  
Application analysis 24  
application class loader 95  
Application Client 34  
application development tool 12  
application packaging

packaging 99  
Application partitioning 17  
Application Server Toolkit 22, 26, 229, 242  
applications 99  
ASF 258  
ASTk 22, 26, 229, 242  
Automatic primary key 92  
AVK 101, 112

## B

BCI 29  
BEA T3 36  
BEA WebLogic Server 8.1 91  
best practices 2, 308, 312  
bootstrap 94  
Bottom Up mapping 55  
Business processes 46  
Byte-code Instrumentation 29

## C

Caching Proxy 13  
class loader 90  
Class loader modes 96  
ClassLoader 93  
ClassNotFoundError 93  
CLASSPATH 94  
ClearCase 24  
Cloudscape 20  
clustering 13  
CMP EJB 2.0 115, 145  
CMP EJBs 55  
CMT 321  
COBOL 26  
Code coverage 29  
coexistence migration 58  
cohesion 303  
Compatibility Test Suite 89  
Concurrent Versions System 24  
Construction 298  
Container Managed Transactions 321  
CORBA 37  
CORBA CSiv2 44  
CORBA Object Transaction Service 44

coupling 303  
CTS 89  
CVS 24

## D

DAO 316  
Data Access Objects 316  
Data Definition Language 268  
Database migration 59  
database schema 97  
Database servers 20  
DDL 59, 268  
Dependency Injection 304, 317  
Deployment Automation 27  
deployment descriptors 97  
design patterns 314  
Developers 298  
Development environment 33  
development tools 22  
DI 317  
Directory servers 21  
DLL 104  
Dynamic operations 15

## E

Edge components 13  
EGL 26  
EIS 2  
EJB CMP 2.0 3  
EJB module 100  
EJB QL 110  
EJB SSB 2.0 3  
EJB to RDB mapping tool 105  
Elaboration 298  
Enhanced EAR 27  
Enterprise Generation Language 26  
Enterprise Information System 2  
Enterprise Java Beans 34  
Enterprise Resource Planning 10  
Enterprise Service Bus 36  
Entity and Stateless Session EJB 2.0 115  
Environments  
    development 33  
    production 33  
    test 33  
ERP 10  
ESB 36  
Execution time analysis 29

Extended manageability 17

## F

Façade 316  
factory design pattern 317  
fix 66  
fixpack 72  
Functional Verification Test 35  
FVT 35

## G

GNU General Public license 262  
graphical user interface 10  
Groovy 4  
GUI 10

## H

Hardware 46  
hardware 64  
Haydes technology 24  
Hibernate 57, 59, 267–268  
Hibernate 2.1.8 4  
high availability 13  
high cohesion 314  
high performance 13  
High-performance computing 16  
HP-UX 12, 19  
HSQL 104, 266–267  
HSQL 1.7.3 4  
HTML 25  
HTTP 10, 36  
HTTP session 320  
HTTP socket 37

## I

i5/OS 12, 19  
IBM DB2 20  
IBM DB2 UDB 8.2 113  
IBM Directory Server 21  
IBM HTTP Server 20  
IBM HTTP Server V2.0.43 20  
IBM HTTP Server V6 20  
IBM Rational Application Developer V6 113  
IBM WebSphere Application Server V6 113  
IDE 51  
IDL 11  
IIOP 37

- Inception 298
- Informix 21, 59
- InitialContext factory 106
- install applications 80
- Integrated Development Environments 51
- Integration with other products 47
- interim fix 66
- Interoperability 36
  - Apache Tomcat 5.5.9 41
  - BEA WebLogic Server 8.1 38
  - Best performance 43
  - Bi-directional 42
  - Client transaction 45
  - Easy to setup 43
  - JBoss 3.2.7 40
  - JTA (2PC) support 44
  - Loose coupling 44
  - Low cost 43
  - Security propagation 44
  - Strategic 43
  - Supported 43
  - Synchronous 42
  - Tested 43
- Inversion of Control 317
- iSeries 20
- iterative model 296
- ivata groupware 3

## J

- J2EE 2, 10–11
- J2EE 1.5 annotations 54
- J2SE 10
- J2SE 1.4 113
- J2SE 1.5 260
- JACL 83
- Jakarta Commons Logging V1.2 96
- Java 10
- Java 1.5 annotations 54
- Java 2 Platform Enterprise Edition 10
- Java 2 Standard Edition 10
- Java Application Verification Kit 89, 101
- Java Application Verification Kit (AVK) for the Enterprise 112
- Java Connector Architecture 11
- Java DataBase Connectivity 11
- Java Interface Definition Language 11
- Java Mail 1.2 3
- Java Message Service 11

- Java Native Interface 104
- Java RMI 36
- Java Server Faces 25, 315, 319
- Java Server Pages 25
- Java Transaction API 11
- JavaDoc 27, 306
- JavaMail 11
- JavaMail API 104
- JAX-RPC 109
- JBoss 3.2.7 90
- JCA 11
- JCL 96
- JDBC 11, 57
- JDBC 2.1 3, 115
- JDO 57
- JDom 96
- JIDL 11
- JMS 11, 36, 106
- JMS 1.0.2 3, 115
- JMS Activation Specification 106
- JNDI Explorer 82
- JNDI names 54
- JNI 104
- JRockit SDK 1.4.2 114
- JSF 25, 315, 319
- JSP 1.2 3, 115, 145
- JSP 1.2 taglibs 3
- JSP Standard Type Library 110
- JSTL 110
- JTA 11, 37
- JTS 11
- JUnit 5, 308
- JUnit technology 24
- JUnitEE 5, 145
- Jython 83

## K

- KeySequenceDirect 92

## L

- Linux (Intel) 19
- Linux (Power PC) 19
- Linux/Intel 12
- Linux/PPC 12
- Load Balancer 13
- Log4j 93
- Log4j.jar 100
- Lotus Domino Enterprise Server 20–21

low coupling 314  
low-level design 306

## M

manage applications 81  
Managers 298  
mapping  
    Bottom Up 55  
    Meet in The Middle 55  
    Top Down 55  
mapping tool 105  
Maven 261  
Maven 1.0.2 4  
Meet in The Middle mapping 55  
memory analysis 28  
memory leak 28  
memory usage 28  
Message Driven Beans 2.0 115, 144  
Microsoft Internet Information Services 20  
Microsoft SQL 59  
Microsoft SQL Server 21  
migration  
    best practices 2  
    issues 2  
    process 2  
    tools 2  
migration activities 45  
Migration alternatives 52  
migration approach  
    coexistence 58  
    offline 58  
    switchover 58  
Modularization 34  
MQ 36  
MVC 314  
MySQL 59, 104  
MySql 267  
MySQL 4.1 4

## N

NanoContainer 1.0 4  
NDS eDirectory 21

## O

Object Oriented 303  
offline migration 58  
OLTP 16

OMG 37  
On demand router architecture 16  
Open Services Infrastructure 11  
OpenSymphony OSCache 271  
OpenSymphony OSCache 1.7.5 5  
OpenSymphony SiteMesh 5  
OpenSymphony WebWork 5  
Operating systems 19  
Oracle 20, 59, 267  
OS/390 12  
OS/400 12, 19  
OSCache 1.7.5 5  
OTS 44

## P

Packaging  
    WebSphere 11  
packaging 13  
PetStore 144, 280  
PicoContainer 318  
PicoContainer 1.2 4  
Plain Old Java Object 268  
planning activities 45  
POJO 262, 268  
port configuration 72  
portability 173  
portable applications 2  
primary key 92  
Primary Key generation 115  
probe editors 29  
probe kit 29  
Production environment 33  
Profiling 28  
Profiling monitor 28  
Prototyping 301

## Q

QoS 37, 42, 258  
Quality 302

## R

Rapid deployment 27  
Rational Application Developer 12  
Rational Application Developer V6 22  
Rational ClearCase 23  
Rational ClearCase LT 24  
Rational Product Updater 66

- Rational RequisitePro 23
- Rational Rose 304
- Rational Tools Suite 23
- Rational Unified Process 7, 23, 297–298
- Rational Web Developer 12, 24
- Rational Web Developer V6 22
- Redbooks Web site 328
  - Contact us xiii
- Reusable components 302
- RMI-IIOP 37
- Roles
  - Analysts 298
  - Developers 298
  - Managers 298
  - Testers 298
- RUP 7, 23, 297
  - Construction 298
  - Elaboration 298
  - Inception 298
  - Transition 298

## S

- SAAJ APIs 109
- Scalability 11
- schema mapping 97
- SCM 24
- SDO 25
- servers
  - database 20
  - directory 21
  - Web 19
- serverStatus 69
- Service Data Objects 25
- Servlet 2.3 3, 115, 145
- ServletContextListener 91
- ServletRequestAttributeListener 110
- ServletRequestListener 110
- Servlets 25
- Session Façade 314
- Simple Object Access Protocol 36
- Singletons 317
- SiteMesh 5
- SMP 17
- SMTP 37
- SOAP 36
- SOAP-HTTP 37
- software development
  - iterative model 296

- waterfall model 296
- Solaris 12
- Source code 46
- Source Code Management 24
- Spring Framework 318
- SQLWays 59
- startServer 69
- Stateful Session Beans 320
- Stateful Session Beans 2.0 144
- Stateless Session Beans 2.0 144
- stopServer 70
- Struts 25, 271, 315, 319
- Struts 1.2.4 4
- Struts applications 25
- Sun Java System Web Server 20
- Sun ONE 20
- Sun ONE Directory Server 21
- Sun Solaris 19
- Supported database servers 20
- Supported directory servers 21
- Supported operating systems 19
- Supported Web servers 20
- SVT 35
- switchover migration 58
- Sybase 21, 59
- symmetric multiprocessor 17
- System Integration Test 35

## T

- T3ShutdownDef 91
- T3StartupDef 91
- tags
  - @ejb 54
  - @web 54
- TCO 15
- Test environments 33
- Testers 298
- Thread analysis 28
- Thread view 28
- time analysis 29
- Timer Service 92
- Tool Mentors 299
- Top Down mapping 55
- Topology 46
- Torque 268
- Trade 3
- Trade 3.1 112
- Transition 298

## U

- UML 23, 303–304
- UML sequence view 28
- UML Topics 23
- Unified Classloader 93
- Unified Modeling Language 23, 303–304
- Unit testing 24, 307
- Universal Test Client 56, 81
- use case 286
- UTC 56
- Utility Classes 34
- utility.jar 100

## V

- Velocity 1.3.1 5

## W

- waterfall model 296
- Web Application 34
- Web application module 100
- Web module class loader 95
- Web servers 19
- Web services 10, 36, 54
- WebSphere 9
  - packaging 11
- WebSphere Application Server 11
  - Requirements 21
- WebSphere Application Server - Express 12
  - Requirements 21
- WebSphere Application Server Network Deployment 13
  - Requirements 21
- WebSphere Application Server V6 - Base 12
- WebSphere Application Server V6 - Express 12
- WebSphere Application Server V6 - Network Deployment 13
- WebSphere Extended Deployment 15
- WebSphere partition facility 17
- WebSphere profiles 73
- WebSphere Rapid Deployment 6, 26, 54, 165, 173
- WebSphere Studio Application Developer 22
- WebSphere Studio Web Developer 24
- WebSphere XD 15
- WebWork 5
- Windows 12, 19
- Windows Active Directory 21
- Windows DLL, 104
- WORA 89

- WRD 6, 51, 53–54, 165, 173
- WRD annotations 53
- Write Once, Run Anywhere 89
- WSAD 22
- wsadmin scripts 83
- WS-AtomicTransaction 44
- WSDL 37
- WS-I Basic Security Profile 44
- WSMQ 36, 44
- WS-Security 44

## X

- Xalan 93
- XD 15
- XDoclet 23, 51, 145, 165
- XDoclet 1.2 4
- XDoclet annotations 56
- XDoclet tags
  - @ejb 54
  - @web 54
- Xerces 93
- XML 10, 25
- xPetstore EJB 3
- xPetstore EJB 3.1.3 112
- xPetstore Servlet 3
- XSLT 49

## Z

- z/OS 19
- z/OS Security Server 21
- zLinux 12, 19



## Migrating Applications from WebLogic, JBoss and Tomcat to WebSphere V6

(0.5" spine)  
0.475" <-> 0.875"  
250 <-> 459 pages









**Redbooks**

# Migrating Applications from WebLogic, JBoss and Tomcat to WebSphere V6

## **Migration strategy and planning**

## **Development and migration tools**

## **Step-by-step instructions**

This IBM Redbook will help you plan and execute the migration of J2EE™ 1.3 applications developed for BEA WebLogic Server 8.1, JBoss 3.2.7 and Apache Tomcat 5.5.9, so that they will run on WebSphere Application Server V6.

This redbook provides detailed information to help you plan migrations, best practices for developing portable applications as well as migration working examples for each of the platforms from which we migrated.

It is not our intention to provide a feature-by-feature comparison of BEA WebLogic Server 8.1, JBoss 3.2.7 and Apache Tomcat 5.5.9 versus WebSphere Application Server V6, nor to argue the relative merits of the products, but to produce practical technical advice for developers who have to migrate applications from these vendors to WebSphere Application Server V6.

## **INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

## **BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)