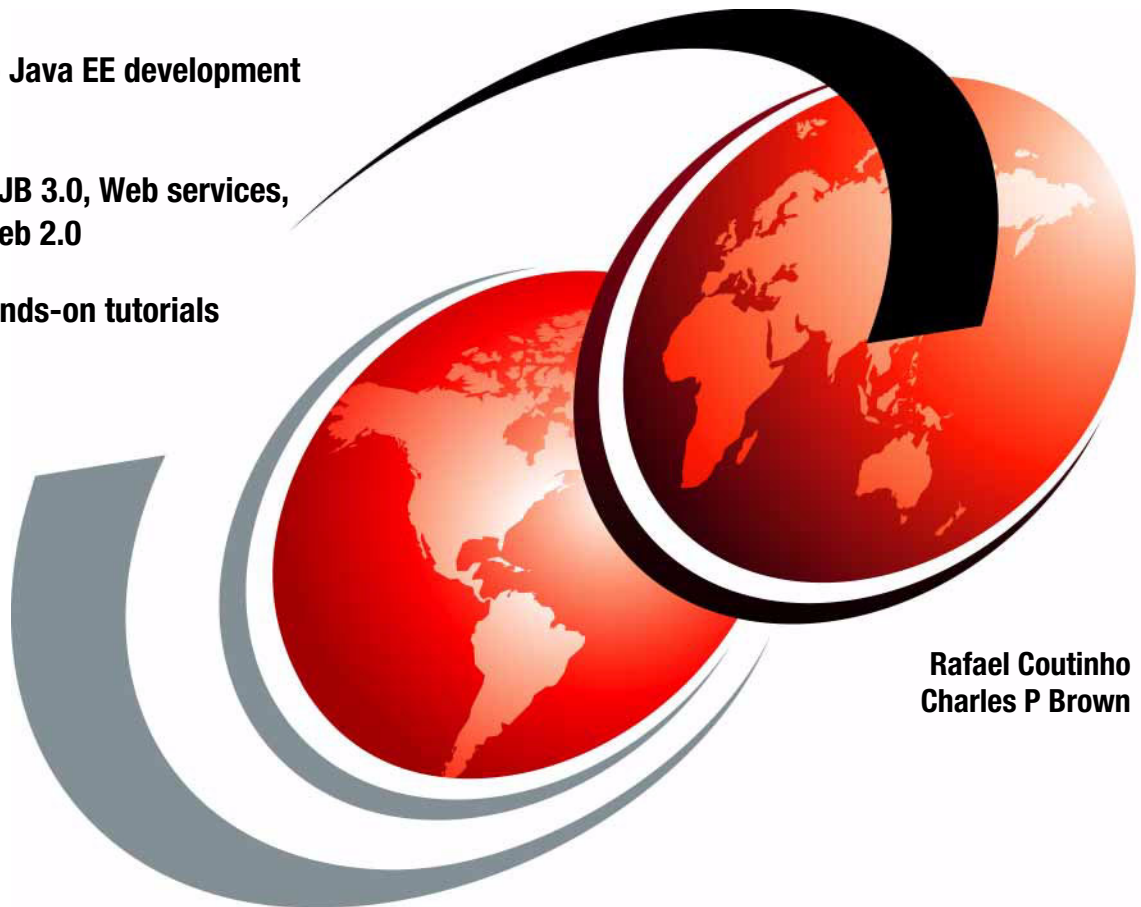


# Experience Java EE! Using Rational Application Developer V7.5

On-ramp to Java EE development

Including EJB 3.0, Web services,  
JMS and Web 2.0

Detailed hands-on tutorials



Rafael Coutinho  
Charles P Brown





International Technical Support Organization

**Experience Java EE! Using Rational Application  
Developer V7.5**

February 2010

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

**First Edition (February 2010)**

This edition applies to Rational Application Developer V7.5 and the embedded WebSphere Application Server V7.0 test environment.

**© Copyright International Business Machines Corporation 2010. All rights reserved.**

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

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

# Trademarks

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

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

AIX®	IBM®	Redbooks (logo)  ®
CICS®	Maximo®	Tivoli®
DataPower®	Passport Advantage®	WebSphere®
DB2®	Rational®	
developerWorks®	Redbooks®	

The following terms are trademarks of other companies:

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Acrobat, Adobe Flash, Adobe Flex, Adobe, Flex Builder, Macromedia, MXML, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

SUSE, the Novell logo, and the N logo are registered trademarks of Novell, Inc. in the United States and other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

SAP NetWeaver, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

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

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

Intel, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo 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.

# Contents

<b>Notices</b> .....	iii
Trademarks .....	iv
<b>Preface</b> .....	xv
The team that wrote this book .....	xvi
Become a published author .....	xvi
Comments welcome .....	xvii
<b>Part 1. Preliminary activities</b> .....	1
<b>Chapter 1. Introduction</b> .....	3
1.1 Learn, Experience, Explore! .....	4
1.2 Java EE simplified .....	7
1.2.1 Objective .....	7
1.2.2 Specifications .....	9
1.2.3 Implementations .....	11
1.2.4 Web services and messaging .....	13
1.2.5 Container architecture .....	14
1.3 Introduction to the scenario .....	19
1.4 Jump start .....	22
1.5 Linux variations .....	23
<b>Chapter 2. Install and configure software</b> .....	25
2.1 Required software .....	26
2.2 Hardware requirements .....	26
2.3 Software requirements .....	27
2.4 Install the software (fastpath) .....	28
2.5 Install the software (advanced) .....	37
2.5.1 Launchpad .....	38
2.5.2 IBM Installation Manager .....	40
2.5.3 Installing IBM Rational Application Developer .....	43
2.5.4 Rational Application Developer updates .....	44
2.5.5 Rational Application Developer License .....	49
<b>Chapter 3. Configure the development environment</b> .....	53
3.1 Learn! .....	54
3.1.1 Eclipse Project .....	54
3.1.2 Workbench and Workspace .....	55

3.2	Extract the base sample code file . . . . .	56
3.3	Create the workspace . . . . .	57
3.4	Create the ExperienceJEE server profile . . . . .	62
3.5	Create the ExperienceJEE workspace test server . . . . .	72
3.6	Start the ExperienceJEE Test Server . . . . .	78
3.7	Import the sample code snippets . . . . .	78
3.8	Workspace preferences . . . . .	80
3.9	Explore! . . . . .	81
<b>Chapter 4.</b>	<b>Prepare the legacy application . . . . .</b>	<b>85</b>
4.1	Learn! . . . . .	86
4.2	Start the Derby Network Server . . . . .	87
4.3	Configure the Derby JDBC driver . . . . .	88
4.4	Create and configure the Vacation database . . . . .	90
4.4.1	Create the Vacation Derby database . . . . .	90
4.4.2	Create the employee table in the Vacation database . . . . .	94
4.4.3	Populate the Vacation database . . . . .	98
4.5	Create the Donate database . . . . .	99
4.6	Configure the ExperienceJEE Test Server . . . . .	100
4.6.1	Start the Administrative Console . . . . .	100
4.6.2	Create the JAAS authentication aliases . . . . .	103
4.6.3	Create the Derby XA JDBC provider . . . . .	109
4.6.4	Create the Vacation JDBC data source . . . . .	112
4.6.5	Create the Donate JDBC data source . . . . .	118
4.7	Explore! . . . . .	120
<b>Part 2.</b>	<b>Core Java EE application . . . . .</b>	<b>121</b>
<b>Chapter 5.</b>	<b>Create the JPA entities . . . . .</b>	<b>123</b>
5.1	Learn! . . . . .	124
5.1.1	Entities . . . . .	125
5.1.2	Mapping the table and columns . . . . .	126
5.1.3	Entity identity . . . . .	129
5.1.4	Callback methods and listeners . . . . .	133
5.1.5	Relationships . . . . .	134
5.1.6	Entity inheritance . . . . .	139
5.1.7	Persistence units . . . . .	141
5.1.8	Object-relational mapping through orm.xml . . . . .	143
5.1.9	Persistence provider . . . . .	144
5.1.10	Entity manager . . . . .	144
5.1.11	Entity life cycle . . . . .	148
5.1.12	JPA query language . . . . .	148



5.2	Jump start . . . . .	153
5.3	Create the database connection . . . . .	153
5.4	Create an enterprise application project . . . . .	154
5.5	Create and configure a JPA project . . . . .	157
5.5.1	Create the DonateJPAEmployee project . . . . .	157
5.5.2	Project dependencies . . . . .	161
5.6	Generate the Employee entity from a table. . . . .	162
5.6.1	Create the Employee entity. . . . .	162
5.6.2	Update the persistence.xml file. . . . .	166
5.7	Create the Fund entity. . . . .	168
5.7.1	Create the DonateJPAFund project . . . . .	168
5.7.2	Create the Fund class . . . . .	169
5.7.3	Configure the Fund entity . . . . .	171
5.7.4	Define the Fund entity database mapping . . . . .	176
5.7.5	Add the Fund entity to the persistence.xml. . . . .	178
5.8	Test the Employee and Donate entities . . . . .	179
5.8.1	Create the JUnit project . . . . .	180
5.8.2	Create the persistence units for the Java SE profile . . . . .	181
5.8.3	Create the JUnit test case. . . . .	184
5.8.4	Run the JUnit test . . . . .	186
5.9	Errors and warnings . . . . .	192
5.10	Explore! . . . . .	196
5.10.1	Advanced topics for persistent entities . . . . .	196
5.10.2	Experience advanced entities with JPA . . . . .	197
<b>Chapter 6.</b>	<b>Create the entity facade session beans . . . . .</b>	<b>201</b>
6.1	Learn! . . . . .	202
6.1.1	EJB 3.0 specification. . . . .	204
6.1.2	EJB 3.0 simplified model. . . . .	204
6.1.3	EJB types and their definition . . . . .	206
6.1.4	Best practices for developing EJBs. . . . .	211
6.1.5	Message-driven beans . . . . .	212
6.1.6	Web services. . . . .	212
6.1.7	Life cycle events . . . . .	212
6.1.8	Interceptors . . . . .	214
6.1.9	Dependency injection . . . . .	216
6.1.10	Using deployment descriptors. . . . .	220
6.1.11	EJB 3.0 application packaging . . . . .	220
6.2	Jump start . . . . .	220
6.3	Create and configure an EJB 3.0 project . . . . .	221
6.3.1	Create the Project. . . . .	221
6.3.2	Add project dependencies. . . . .	224

6.4	Create the entity facade interfaces and session beans. . . . .	225
6.4.1	Define the business interface for the employee facade . . . . .	225
6.4.2	Define the remote interface for the employee facade . . . . .	227
6.4.3	Create the session bean facade for the employee entity . . . . .	228
6.4.4	Define the business interface for the fund facade . . . . .	234
6.4.5	Define the remote interface for the fund facade . . . . .	235
6.4.6	Create the session bean facade for the fund entity. . . . .	235
6.5	Deploy the DonateEAR enterprise application . . . . .	240
6.6	Test the session facades with the Universal Test Client . . . . .	243
6.6.1	Start the UTC . . . . .	244
6.6.2	Test the EmployeeFacade EJB with the UTC . . . . .	244
6.7	Create the EJB 3.0 stub classes . . . . .	248
6.7.1	Create the build.xml ANT script . . . . .	248
6.7.2	Create the build.xml ANT run configuration . . . . .	250
6.7.3	Execute the build.xml ANT run configuration . . . . .	252
6.7.4	Add the EJB stubs to the DonateUnitTester build path. . . . .	253
6.8	Test the session facades with JUnit . . . . .	254
6.8.1	Create the employee facade tester . . . . .	254
6.8.2	Run the employee facade tester . . . . .	256
6.8.3	Create the fund facade tester . . . . .	258
6.8.4	Run the fund facade tester . . . . .	258
6.9	Verify the databases . . . . .	259
6.10	Explore! . . . . .	259
<b>Chapter 7.</b>	<b>Create the Donate session bean for the business logic . . . .</b>	<b>261</b>
7.1	Learn! . . . . .	262
7.2	Jump start . . . . .	262
7.3	Define the Donate session EJB. . . . .	263
7.3.1	Define the business interface for the Donate session bean . . . . .	263
7.3.2	Define the remote interface for the Donate session bean . . . . .	263
7.3.3	Create the Donate session bean. . . . .	265
7.4	Test the Donate session EJB with the UTC . . . . .	268
7.5	Test the session bean with JUnit. . . . .	272
7.5.1	Update the EJB stubs . . . . .	272
7.5.2	Create the Donate bean test case . . . . .	272
7.5.3	Run the Donate bean tester . . . . .	274
7.6	Explore! . . . . .	275
<b>Chapter 8.</b>	<b>Create the Web front end . . . . .</b>	<b>277</b>
8.1	Learn! . . . . .	278
8.1.1	What is a servlet? . . . . .	279
8.1.2	What is a JSP? . . . . .	280
8.1.3	Session store . . . . .	280

8.1.4 Controller . . . . .	280
8.1.5 Model-view-controller . . . . .	281
8.1.6 What is a JSF? . . . . .	282
8.1.7 JSF application architecture . . . . .	282
8.1.8 JSF life cycle . . . . .	284
8.1.9 Goals of JSF . . . . .	285
8.2 Jump start . . . . .	286
8.3 Basic Web application . . . . .	286
8.3.1 Create and configure the Web project . . . . .	287
8.3.2 Add the dependency to the EJB project . . . . .	288
8.3.3 Create the JSP . . . . .	289
8.3.4 Create the servlet . . . . .	292
8.3.5 Test the Web application . . . . .	295
8.3.6 Recap . . . . .	297
8.4 JSF Web application . . . . .	297
8.4.1 Suppress page code generation . . . . .	298
8.4.2 Create the JSF Web project . . . . .	299
8.4.3 Create the JSPs . . . . .	302
8.4.4 Define the navigation . . . . .	303
8.5 Create the managed beans . . . . .	310
8.5.1 Define the managed beans . . . . .	310
8.5.2 Add the dependency to the EJB project . . . . .	313
8.5.3 Implement the FindEmployeeManagedBean class . . . . .	313
8.5.4 Implement the DonateHoursManagedBean class . . . . .	315
8.6 Implement the JSP pages . . . . .	317
8.6.1 Implement the FindEmployee page . . . . .	318
8.6.2 Bind FindEmployee to a managed bean . . . . .	323
8.6.3 Implement the EmployeeDetails page . . . . .	327
8.6.4 Implement the DonationConfirmation page . . . . .	334
8.6.5 Recapitulation . . . . .	336
8.7 Test the JSF Web application . . . . .	336
8.8 Optional extensions to DonateJSFWeb . . . . .	339
8.8.1 Error handling . . . . .	340
8.8.2 Validation . . . . .	342

8.9 Explore! JSF .....	344
<b>Chapter 9. Create the application client</b> .....	345
9.1 Learn! .....	346
9.2 Jump start .....	347
9.3 Create the application client project .....	348
9.4 Configure the application client project .....	348
9.5 Code the main method .....	349
9.6 Test the application client .....	350
9.7 Create and test a standalone client. ....	356
9.8 Explore! .....	359
<b>Chapter 10. Implement core security</b> .....	361
10.1 Learn! .....	362
10.1.1 User registry .....	363
10.1.2 Application role-based security .....	363
10.1.3 Administrative security .....	364
10.2 Jump start .....	364
10.3 Preliminary security setup .....	365
10.3.1 Create and configure the Jython project and script. ....	365
10.3.2 Run the Jython script .....	366
10.4 Enable Java EE application security .....	370
10.4.1 Enable Java EE application security .....	370
10.4.2 Restart the ExperienceJEE Test Server .....	373
10.5 Implement declarative security .....	373
10.5.1 Restrict access to the Donate session EJB (declarative) .....	373
10.5.2 Restrict access to Web pages via declarative security .....	375
10.5.3 Configure DonateEAR roles and role mapping .....	380
10.5.4 Test the EJB declarative security using JUnit .....	383
10.5.5 Test the EJB declarative security using the UTC .....	387
10.5.6 Test the Web declarative security .....	389
10.6 Implement and test programmatic security .....	393
10.6.1 Add clone method to the Employee entity. ....	393
10.6.2 Implement programmatic security in the EmployeeFacade ....	394
10.6.3 Test programmatic security in a session EJB .....	396
10.6.4 Implement programmatic security in a Web application .....	397
10.6.5 Test programmatic security in a Web application .....	400
10.7 Update the Java EE application client for security .....	401
10.7.1 Demonstrate the existing behavior .....	402
10.7.2 Create a customized properties file. ....	403
10.7.3 Update and test the Java EE application client test configuration	403
10.8 Explore! .....	404
10.8.1 Java EE Security .....	404

<b>Part 3. Web services</b> .....	407
<b>Chapter 11. .... Create the Web service</b>	409
11.1 Learn! .....	410
11.1.1 Web services and Java EE 5 .....	412
11.2 Jump start .....	414
11.3 Disable Java EE EJB declarative security .....	414
11.4 Create the bottom-up EJB Web service .....	416
11.5 Test the Web service using the Web Services Explorer .....	425
11.6 Create and test a Web service client .....	429
11.6.1 Create and test the Web service client .....	429
11.7 Optional .....	436
11.7.1 Monitor the SOAP traffic using TCP/IP Monitor .....	437
11.7.2 Create a client servlet .....	441
11.7.3 Run the client servlet .....	443
11.8 Explore! .....	444
11.8.1 Handlers in JAX-WS .....	444
11.8.2 Asynchronous Web service invocation .....	445
11.8.3 Attachments performance improvement .....	445
11.8.4 Creating a top-down Web service from a WSDL .....	446
<b>Chapter 12. Implement security for the Web service</b> .....	447
12.1 Learn! .....	448
12.2 Jump start .....	452
12.3 Re-enable Java EE EJB declarative security .....	452
12.4 Test the Web service as-is .....	453
12.5 Update the Web service to support secure access .....	454
12.5.1 Create the policy set .....	454
12.5.2 Create the client policy set bindings .....	458
12.5.3 Create the provider policy set bindings .....	461
12.5.4 Import the policy set and binding files .....	463
12.5.5 Attach the policy set to the Web service provider .....	465
12.5.6 Attach the policy set to the Web service client .....	466
12.5.7 Test the secure Web service using the DonateWSCClient .....	467
12.6 Explore! .....	470
12.6.1 Additional resources .....	471
12.6.2 Secure Web services using transport level security .....	471
12.6.3 Secure Web services with DataPower .....	472
<b>Part 4. Messaging</b> .....	475
<b>Chapter 13. Create the message-driven bean</b> .....	477
13.1 Learn! .....	478
13.1.1 Additional resources .....	480

13.1.2	EJB 3.0 message-driven beans	481
13.1.3	Service integration bus	482
13.2	Jump start	483
13.3	Disable Java EE EJB declarative security	483
13.4	Configure the test server for base JMS resources	484
13.4.1	Create the Service Integration Bus	484
13.4.2	Configure security	490
13.4.3	Create the destination queue resource	493
13.4.4	Create the JMS queue destination	496
13.4.5	Create the JMS activation specification	498
13.4.6	Create the JMS connection factory for the application client	500
13.4.7	Restart the ExperienceJEE Test Server	502
13.5	Create the message-driven bean	502
13.5.1	Define the MDB	502
13.5.2	Add the activation specification information	504
13.5.3	Code the DonateMDB message-driven bean	506
13.6	Create the messaging client (DonateMDBClient)	508
13.7	Test the messaging client	514
13.8	Explore!	515
13.8.1	Integration with WebSphere MQ	516
<b>Chapter 14.</b>	<b>Add publication of results</b>	<b>517</b>
14.1	Learn!	518
14.2	Jump start	519
14.3	Update the DonateBean session EJB	520
14.4	Create the messaging pub sub client (DonatePubSubClient)	525
14.5	Test the messaging pub sub client	528
14.5.1	Start DonatePubSubClient	528
14.5.2	Submit requests	530
14.6	Explore!	531
<b>Chapter 15.</b>	<b>Implement security for messaging</b>	<b>535</b>
15.1	Learn!	536
15.2	Jump start	537
15.3	Re-enable Java EE EJB declarative security	538
15.4	Test the messaging artifacts as-is	539
15.5	Update the DonateMDBClient and DonateMDB	539
15.5.1	Update the DonateMDBClient code	539
15.5.2	Update the DonateMDB code	540

15.6 Test the updated messaging artifacts . . . . .	542
15.7 Explore! . . . . .	543
<b>Part 5. Advanced Java EE 5 and Web 2.0 . . . . .</b>	<b>545</b>
<b>Chapter 16. Develop a Web 2.0 client . . . . .</b>	<b>547</b>
16.1 Learn . . . . .	548
16.1.1 What is a Web 2.0 client? . . . . .	548
16.1.2 Technologies for Web 2.0 client development . . . . .	549
16.1.3 WebSphere Feature Pack for Web 2.0 . . . . .	553
16.2 Web 2.0 scenario use cases . . . . .	558
16.2.1 Use case requirements . . . . .	558
16.2.2 Rich user interface requirements . . . . .	560
16.3 Installing the Feature Pack for Web 2.0 v1.0.0.1 . . . . .	561
16.4 Jump start . . . . .	564
16.5 Disable Java EE EJB application security. . . . .	564
16.6 Setup the development environment. . . . .	565
16.6.1 Create a Dynamic Web project . . . . .	566
16.6.2 Enable the web 2.0 facets. . . . .	566
16.6.3 Add the dependency to the EJB project . . . . .	569
16.6.4 Populate the Fund table . . . . .	569
16.7 Implement the Find Employee use case . . . . .	570
16.7.1 Create a REST servlet . . . . .	570
16.7.2 Test the FindEmployee REST servlet. . . . .	575
16.7.3 Create the Find Employee view . . . . .	576
16.7.4 Test the Find Employee Use Case . . . . .	582
16.8 Add style to the Web 2.0 application. . . . .	583
16.8.1 Create a cascading style sheet. . . . .	584
16.8.2 Copy the images . . . . .	588
16.8.3 Add style definitions to the HTML files . . . . .	588
16.8.4 Test the Find Employee use case with style . . . . .	590
16.9 Implement the Donate Vacation use case. . . . .	591
16.9.1 Create the view for vacation donation. . . . .	591
16.9.2 Implement the client logic for the Donate Vacation use case . . . . .	595
16.9.3 Display messages in Ajax modal dialogs . . . . .	598
16.9.4 Create a REST servlet for the Donate Vacation use case . . . . .	599
16.9.5 Test the Donate Vacation use case . . . . .	601
16.9.6 Update the vacation hours automatically . . . . .	604
16.9.7 Retrieve the fund names from the database. . . . .	606
16.10 Monitor the Web 2.0 interactions. . . . .	609
16.11 Explore . . . . .	610
16.11.1 Flex . . . . .	610
16.11.2 Macromedia XML (MXML) . . . . .	610

16.11.3 ActionScript . . . . .	611
<b>Part 6. Appendixes . . . . .</b>	<b>613</b>
<b>Appendix A. Jump start . . . . .</b>	<b>615</b>
A.1 Common jump start activities . . . . .	616
A.2 Chapter specific jump start steps . . . . .	618
<b>Appendix B. Additional material . . . . .</b>	<b>629</b>
Locating the Web material . . . . .	630
Using the Web material . . . . .	630
System requirements for downloading the Web material . . . . .	630
How to use the Web material . . . . .	630
<b>Abbreviations and acronyms . . . . .</b>	<b>631</b>
<b>Related publications . . . . .</b>	<b>633</b>
IBM Redbooks . . . . .	633
Online resources . . . . .	634
How to get IBM Redbooks publications . . . . .	635
Help from IBM . . . . .	635



# Preface

This IBM® Redbooks® publication is a hands-on guide to developing a comprehensive Java™ EE application using Rational® Application Developer V7.5 and the embedded WebSphere® Application Server V7.0 test environment, including items such as core functions, security, Web services, and messaging.

Novice users are thus able to experience Java EE, and advance from theoretical knowledge gained by reading introductory material to practical knowledge gained by implementing a real-life application.

*Experience* is one stage in gaining and applying knowledge, but there are additional stages needed to complete the knowledge acquisition cycle. This book also helps in those stages:

- ▶ Before experiencing Java EE, you *learn* about the base specifications and intellectual knowledge of Java EE through brief descriptions of the theory and through links to other information sources.
- ▶ After experiencing Java EE, you *explore* advanced Java EE through previews of advanced topics and links to other information sources.

The target audience is technical users who have minimal experience with Java EE and the IBM WebSphere Application Server product set, but who do have past experience in using an integrated development environment (IDE) and in architecting or designing enterprise applications.

## The team that wrote this book

This book was adapted from earlier redbooks (*Experience J2EE! using WebSphere Application Server v6.1* and *Experience Java EE! using WebSphere Community Edition 2.1*) by the following team:

**Charles P Brown** (Charlie Brown) is an IBM Executive IT Specialist in the WebSphere brand of products. He spent over 25 years with IBM working in robotics, software development, beta support programs (for IBM in the United Kingdom), and for the last six years supporting IBM partners. Charlie's focus is the practical application of product and technology, helping partners and clients develop skills, and then assisting them in being successful in their initial implementations. Charlie holds a Bachelors of Science in Mechanical Engineering (M.I.T., 1984) and a Masters of Science in Computer Science (Johns Hopkins, 1989). Charlie was the author of the original Redbook in this series (*Experience J2EE! using WebSphere Application Server v6.1*!).

**Rafael Coutinho** is an IBM Software Engineer working for Software Group in the Brazil Software Development Lab. His professional expertise covers many technology areas ranging from embedded to platform based solutions. He is currently working on the geographic information system (GIS) add-on of IBM Maximo® Enterprise Asset Management (EAM). He is a certified Java enterprise architect and Accredited IT Specialist, specialized in high-performance distributed applications on corporate and financial projects. Rafael is a computer engineer graduate from the State University of Campinas (Unicamp), Brazil, and has a degree in information technologies from the Centrale Lyon (ECL), France.

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

- ▶ Carla Stadtler, IT Specialist, IBM Raleigh (ITSO project leader)
- ▶ LindaMay Patterson, Information Developer, IBM (ITSO editor)
- ▶ Rodrigo de Oliveira Murta, IT Specialist, IBM Brazil (reviewer of draft redbook)
- ▶ Chris Day, Technical Pre-Sales Specialist, Meier Business Systems Pty Ltd, Sydney, Australia (reviewer of draft redbook)
- ▶ Ueli Wahli, Steven Calello, Patrick Gan, Cedric Hurst, and Maan Mehta (co-authors of Redbook *Experience Java EE! using WebSphere Community Edition 2.1*)

## Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with

leading-edge technologies. You can have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts can help increase product acceptance and customer satisfaction. As a bonus, you can 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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

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

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

- ▶ Send your comments in an e-mail to:

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

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099, 2455 South Road  
Poughkeepsie, NY 12601-5400





# Part 1

# Preliminary activities

In Part 1, we provide introduction to both Java EE and the scenario used in this book. Also, in Part 1, we show the following activities:

- ▶ Installing and configuring the prerequisite software
- ▶ Configuring the development environment
- ▶ Creating and populating the database that represents the legacy application





# Introduction

In this chapter, we introduce you to the following items:

- ▶ The purpose and intent of this IBM Redbooks publication
- ▶ An overall view of Java EE
- ▶ The scenario that forms the basis for the remainder of this publication

# 1.1 Learn, Experience, Explore!

*“We have developers who know C++ and PERL or Microsoft® .NET, but we have to get them up to speed quickly on Java EE. Is there a good introductory hands-on book that provides a cohesive view of Java EE with enough information to allow them to develop a real-life application?*

*Is there a book that helps them to experience Java EE?”*

This same basic question has been voiced over many technologies in the Information Technology (IT) industry, most recently with Java Platform, Enterprise Edition (Java EE) 5. Refer to 1.2, “Java EE simplified” on page 7 for a brief introduction to Java EE.

Java EE developers have access to the many Web sites, books, articles, and news groups that deal with Java EE and related technologies. Still, new Java EE developers find it very difficult to move from reading this mass of information to actually applying the technology to solve business issues.

So, how do developers effectively use *Java EE to get something done*? And what is the optimal way to obtain the knowledge they need?

The two previous IBM Redbooks publications, *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297, and *Experience Java EE! Using WebSphere Application Server Community Edition 2.1*, SG24-7639, suggested that the process of acquiring and applying knowledge of a technology would be more effective if the learning process was separated into the following three phases:

- ▶ **Learn**                      The basic specifications, intellectual knowledge, and theory of the technology, through reading specifications and product documentation.
- ▶ **Experience**                A simple but still real-life application of the technology, through classes, initial usage, and hands-on tutorials.
- ▶ **Explore**                    Advanced features, latest changes, and best practices, through reading articles, specifications, and bulletin boards.

Those books suggested that generally available information about technology is lacking in the Experience phase. Most guided tutorials/examples are written independently to focus on a specific area. This can be helpful for someone who already knows other aspects of the technology, but it can be very frustrating for those trying to experience the overall technology. Because of this approach:



- ▶ The reader is left with disconnected segments of knowledge that they are unable to apply together to develop complete cohesive enterprise applications. For example, after completing a tutorial on session Enterprise JavaBean (EJB) security, the reader might understand that subject area, but does not understand how it links with security for Web front ends, thick clients, Web services, and messaging (JMS).
- ▶ The reader must perform repetitive basic configuration tasks in each scenario (because each guided tutorial/example uses a separate sample application). This increases the amount of time they must invest to learn the technology, and it detracts from the focus and clarity of the example (for example, what is being done as the basic configuration versus what is being done as part of learning the specific scenario).
- ▶ The reader is presented with as much detail as possible about the specific Java EE aspect. This works well for experienced Java EE developers, who use the material as a reference, but it can overwhelm and confuse the Java EE novice.

### Experience UNIX®?

This problem is not unique to Java EE. Step back to the late 1980s and early 1990s and consider the desktop operating system battle that Microsoft Windows® ultimately won. The UNIX operating systems and associated technologies (xWindows, Motif, NFS, Yellow Pages, and others) in the late 1980s contained functionality far superior to the various versions of Microsoft Windows, yet they lost. Why?

These products and their associated user communities were deficient in the Learn and Experience phases, and they did not attempt to cultivate new users. Instead, they focused on arcane knowledge, such as complex keystroke macros for the **vi** editor and cryptic commands such as **awk**, **sed**, and **grep**. The unusability was adopted as a badge of honor and defined the technology and the community.

Ultimately, the UNIX community was not completely oblivious to these issues, as evidenced by the focus on Linux®, providing a user-friendly version of the UNIX operating system for the desktop.

This book is the successor to these two earlier books (SG24-7297 and SG24-7639), enabling readers to experience the full spectrum of basic Java EE application development in a single document using WebSphere Application Server v7.0 and Rational Application Developer 7.5:

- ▶ This publication utilizes a single-integrated scenario across all of Java EE. You learn and set up a single application, and sequentially add the various

Java EE technologies, thus you can focus only on the new technology in each section (and how it relates to the technologies covered in previous sections).

Section 1.3, “Introduction to the scenario” on page 19 describes the scenario used in this book. The following IBM software products are used to implement the Java EE application (Table 1-1).

Table 1-1 IBM software products used to implement the Java EE application

Java EE	Software products
Java EE development tool	Rational Application Developer v7.5 (referred to as the workbench)
Java EE runtime	WebSphere Application Server v7.0 (provided as part of Rational Application Developer V7.5) ► Referred to as the application server.

- This book is kept to a reasonable length by providing links to other information sources for both beginning knowledge (*learning*) and advanced topics (*exploring*), thus avoiding overloading the user with useful but not directly relevant information. Following are some key Web sites referenced in this book:
  - IBM developerWorks:  
<http://www.ibm.com/developerworks>
  - IBM Redbooks:  
<http://www.redbooks.ibm.com>
  - Sun Java EE (J2EE):  
<http://java.sun.com/javaee>

The target readers are technical users, who have minimal experience with the Java EE product set, but who do have past experience in using an integrated development environment (IDE) and in architecting or designing enterprise applications.

You will not become a Java EE expert by reading and experiencing this book, but afterwards you should have enough knowledge for the following tasks:

- Develop, test, and deploy simple Java EE applications that contain necessary qualities of service such as two-phase commit support and a consistent security model.
- Know where to look for additional information (both to learn and explore).

- Phrase the relevant questions that can help you expand your knowledge and apply the appropriate technology to business scenarios.

Use this IBM Redbooks publication as a self-guided tutorial, as demonstration scripts, or as hands-on labs for a workshop. We hope this book leads you to a successful Java EE experience!

## 1.2 Java EE simplified

***“Java Platform, Enterprise Edition (Java EE):** An environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The Java EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multi-tiered, Web-based applications. (Sun)”*

From the WebSphere Application Server Information Center glossary:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.help.glossary.doc/topics/glossary.html>

The formal definition of Java EE is managed by the Java Community Process and is called *JSR 244: Java™ Platform, Enterprise Edition 5 (Java EE 5) Specification*. The current specification is available at:

<http://jcp.org/en/jsr/detail?id=244>

The key point to remember is that Java EE allows developers to create applications using commercial off the shelf (COTS) development software, and then provides these applications to others to install and execute on COTS runtime software.

### 1.2.1 Objective

The Java EE objective is to allow developers to create applications that can be executed—without rewriting or recompiling—on runtime software products provided by a wide variety of software companies across a wide variety of operating systems and services (such as databases and back-end applications).

Java EE acts as the mediator between the application, the operating system, and the back-end services. Java EE masks the differences among the various operating systems and back-end services, providing a consistent set of interfaces and functions to the application. This masking allows a single version

of the source code to support all environments. The term that Sun uses is Write Once, Run Anywhere (WORA).

Java EE, as indicated by the name, is implemented on the Java programming language. Java uses a bytecode compilation strategy, which is a hybrid between the pure interpretive languages, such as Basic, and pure compilation languages, such as C and Pascal. The Java compiler converts the source code into an intermediate bytecode format. This bytecode can then be executed by any operating system that has the Java Virtual Machine (JVM) runtime environment installed. The complete specification is called Java 2 Standard Edition (J2SE). The term that many in the industry use is CORA (Compile Once, Run Anywhere).

In theory, a developer could write an application and, without changing a single line of code, run it on a wide range of environments:

- ▶ Java EE runtime providers (such as, IBM, BEA, Oracle, Sun)
- ▶ Operating systems (such as, Microsoft Windows, Unix, Linux, AIX®)
- ▶ Databases (such as, DB2®, Oracle Database, Sybase, Microsoft SQL Server)
- ▶ Back-end information systems (such as, SAP, PeopleSoft)

In practice, Java EE comes pretty close to meeting this objective. In some situations, applications run on different runtime providers (and operating systems, databases, back-end services) without changing a single line of code or packaging information. WORA and CORA! However, in the real world, most applications require minor adjustments to run in the various environments:

- ▶ Java EE specifications do not cover all technologies and functions, so developers are often forced to write to environment specific interfaces/functions that change when the application is moved to another environment.
- ▶ Java EE specifications are open to interpretations (in spite of the best intentions of those who define the specifications) leading to differences in the application program interfaces and packaging supported by the various Java EE runtime providers.
- ▶ The Java EE runtime cannot completely mask the differences between the various environments. For example, DB2 and Oracle databases have different requirements for table names.

We are not discrediting Java EE by asserting that it cannot support 100% code portability in all situations. Rather, we are trying to set realistic expectations.

Our experiences suggest that most Java EE applications exceed 99% code portability, that is minimal tweaking is needed when moving from one environment to another. This is certainly far better than writing applications

directly to proprietary, platform specific interfaces that mandate almost total code re-write when moving to another environment.

## 1.2.2 Specifications

Java EE meets this objective by providing a set of architectural specifications and features that define and support the relationships between the application, the runtime environments, and the common services.

Java EE contains the following four core components for your use:

- ▶ Development specifications that support the creation of software applications, in particular application programming interfaces (API) and packaging requirements.
- ▶ Execution specifications that define the environments for executing Java EE applications, in particular API implementations, common services, and how to process the software application packaging.
- ▶ Compatibility test suite for verifying that the runtime environment product complies with the Java EE platform standard.
- ▶ Reference implementation that provides a rudimentary implementation of the Java EE specifications. This approach is quite adequate for base learning but does not have extended quality of services such as an administration GUI, logging, clustering, and development tools.

### Changes in Java EE standards

The Java EE standards were first formally released in 1999 and continue to evolve (see Table 1-2).

*Table 1-2 Java EE release information*

Java EE version	Based on J2SE	Final specification release date	JSR (see discussion)
2 (1.2)	1.3	12/1999	n/a
3 (1.3)	1.3	09/2001	n/a
4 (1.4)	1.4	11/2003	151
5	5	05/2006	244
6	6	draft 07/2007	316

Note that the naming structure changed with the fifth version from Java 2 Platform, Enterprise Edition (J2EE) v1.x to Java Platform, Enterprise Edition (Java EE) x. For example, the version 4 specification is formally called J2EE 1.4 (not Java EE 4), however, the version 5 specification is called Java EE 5.

The pace of releases slowed down as the specifications matured, reflecting the broader install base (pressures on maintaining compatibility) and the much broader user community (the more people involved taking time to reach a consensus).

The specification updates are also changing in focus. Earlier Java EE releases added new functions and features, but many of the changes in Java EE 5 are driven from the desire for simplification. For example, one of the key changes is reducing the number of text descriptor files (so called *deployment descriptors*) by moving the information into *annotations* contained directly in the Java files. This reduces the number of overall artifacts and helps avoid the problem of having deployment descriptors out of sync with the Java files.

This switch of emphasis to simplification illustrates the maturity of the Java EE platform. The platform provides enough base functions for users to develop highly-functional and portable applications. Users seem to be generally satisfied with the current functions, and seem to be looking at how it can be done better and simpler.

Sun maintains these specifications, and updates them using a process called the Java Community Process (JCP). The JCP Web site is at:

<http://jcp.org>

JCP works on a broad range of specifications called Java Specification Requests (JSRs). These JSRs might or might not be incorporated into new versions of Java EE and J2SE, based on input from the open community.

The Java EE 5 specification is formally called *JSR 244: Java™ Platform, Enterprise Edition 5 (Java EE 5) Specification*, by Sun, and is available at the following Web site:

<http://jcp.org/en/jsr/detail?id=244>

### Deployment descriptors versus annotations:

For example, an EJB session bean can be defined using an annotation or a deployment descriptor:

- The annotation is practical for programmers:

```
@Stateless
public class FundFacade implements FundFacadeInterface { .... }
```

- The deployment descriptor is practical for deployers:

```
<?xml version="1.0".....>
  <display-name>FundFacade</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>FundFacade</ejb-name>
      <ejb-class>donate.ejb.impl.FundFacade</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

In practice, many robust Java EE application will include both.

## 1.2.3 Implementations

Software companies use these specifications to create and sell development and runtime Java EE software products.

Java EE is owned by Sun Microsystems and licensed to companies that must pass the included compatibility test suite before shipping a product with the Java EE logo.

For a list of current authorized licensees of the Java EE and J2EE platforms, visit the following Web site:

<http://java.sun.com/j2ee/licensees.html>

These companies have permission to ship products that are branded Java EE and have passed the Java EE compatibility tests.

For the list of authorized licencees who have developed products that have passed the Java EE compatibility tests, visit the following Web site:

<http://java.sun.com/javaee/overview/compatibility.jsp>

Table 1-3 lists the current Java EE product providers (commercial and open source) as of the writing of this book:

*Table 1-3 Java EE product providers*

Vendor	Runtime product
Apache	Apache Geronimo v2.1.2
BEA	WebLogic Server 10.0
IBM	WebSphere Application Server Community Edition v2.1 WebSphere Application Server v7.0
Kingdee	Apusic Application Server v5.0
NEC	WebOTX Application Server v8.1
Oracle	Oracle Application Server v11
SAP	SAP NetWeaver v7.1
Sun	Sun Java System Application Server Platform Edition v9
TmaxSoft	TmaxSoft JEUS v6

In theory, an application generated by any development product can run on any runtime product. In reality, there is some affinity between development products and runtime products due to the following:

- ▶ Some of the minor variations in Java EE implementations, due to the specifications being incomplete or imprecise.
- ▶ Support for areas where the runtime product implements proprietary (but still valuable) extensions to the specifications.
- ▶ Choosing to implement draft or up level specifications that are required by the next level higher of Java EE.
- ▶ Testing and debugging tools for that particular runtime product.



### **WebSphere Application Server v7.0 and Rational Application Developer v7.5**

IBM markets two J2EE/Java EE platform platforms, WebSphere Community Edition and WebSphere Application Server.

WebSphere Application Server is the IBM premier Java EE runtime that provides all the programming model specifications plus additional infrastructure support (scalability, administration, failover, logging, diagnostics, clustering) needed to support highly available, highly reliable solutions. WebSphere Application Server is a fee-based product, both for the initial purchase and for support after the first year of ownership (the first year is included in the purchase fee).

In contrast, WebSphere Application Server Community edition has a zero cost purchase, but support is fee-based.

The recently announced WebSphere Application Server v7.0 also supports Java EE 5, and a matching development product, Rational Application Developer v7.5, was announced at the same time. Both products are available as trial downloads at these addresses:

[http://www.ibm.com/developerworks/downloads/ws/was/?S\\_TACT=105AGX28&S\\_CMP=DLMAIN&S\\_CMP=rnav](http://www.ibm.com/developerworks/downloads/ws/was/?S_TACT=105AGX28&S_CMP=DLMAIN&S_CMP=rnav)  
[http://www.ibm.com/developerworks/downloads/r/rad/?S\\_TACT=105AGX23&S\\_CMP=DWNL&S\\_CMP=rnav](http://www.ibm.com/developerworks/downloads/r/rad/?S_TACT=105AGX23&S_CMP=DWNL&S_CMP=rnav)

## **1.2.4 Web services and messaging**

Java EE provides both traditional Internet server and program-to-program interaction through Web services and messaging.

Java EE originated as a set of specifications to support dynamic interaction over hypertext transfer protocol (HTTP) between browsers, thick clients, and back-end systems such as databases. The typical *page* content was and is hypertext markup language (HTML), but the environment supports other formats, such as cHTML, VXML, and XML, as well.

Over time, Java EE continues to refine and extend this support, but it has also been extended to support program-to-program type interaction as well. The two key technologies are Web services (discussed in Part 3, “Web services” on page 407) and messaging (discussed in Part 4, “Messaging” on page 475).

## 1.2.5 Container architecture

The Java EE runtime environment consists of a series of containers that provide a set of common services that mediate between the application components and the actual runtime environment (operating system, database).

The developer can focus on the business logic or presentation interface, and the Java EE container manages the common services such as security, session state, transactions, database access, fail-over, and recovery.

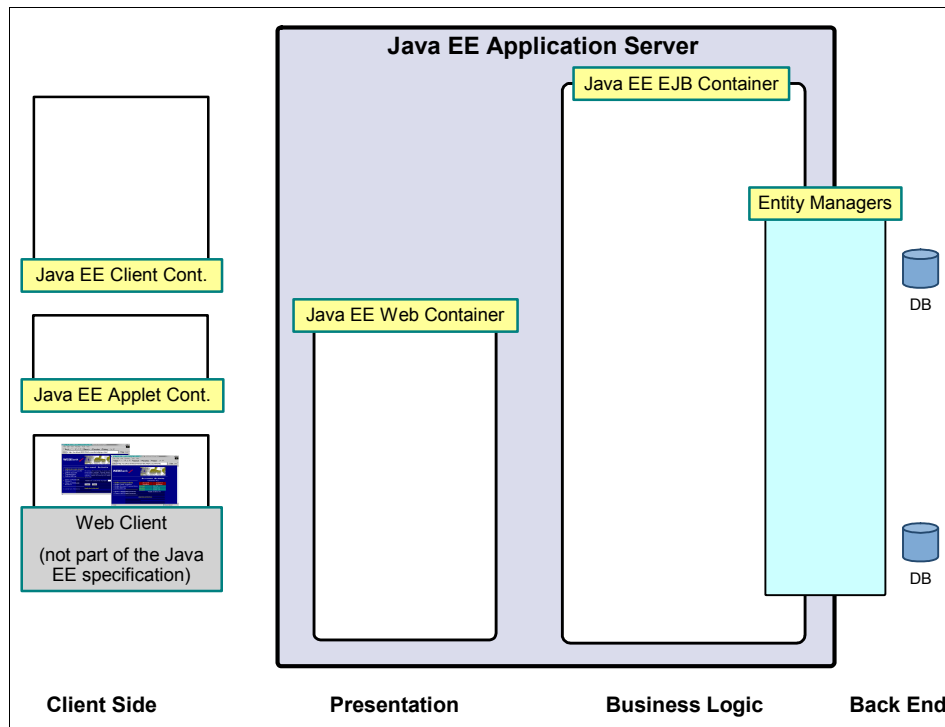


Figure 1-1 Java EE container architecture

The description of Figure 1-1 is meant to provide a brief introduction to the overall Java EE container environment and programming artifacts. The individual scenario chapters will provide more detail on the specific Java EE components (for example, servlets, JSPs, EJBs).

Java EE literature tends to describe the containers in the context of tiers or layers, where each tier or layer performs a specific function in the overall

distributed application. We start the following description of Figure 1-1 on page 14 starting from the left.

## Client side tier

The client side tier represents the *glass* interaction with the user:

- ▶ The Web client is a browser: An application on a client system that supports interaction over HTTP, typically with a content type of HTML. This is a thin client: The user does not have to install any software on the client system (beyond the Web browser).

Technically, this is not a Java EE container and the internal operations and capabilities of the browser are not part of the Java EE specifications and do not have access to the server side Java EE services.

The only specific Java EE requirement is that the browser support HTML 3.2 or higher.

- ▶ The Java EE applet container supports the client side execution of Java programs in the context of a browser session. However, these Java programs have no direct access to server side Java EE services. Applets and client side scripting such as JavaScript have the same capabilities as they would in a non-Java EE environment.
- ▶ The Java EE Client container supports the client side execution of Java programs (for example, a Java class file with a main method) that have access to server side Java EE services, such as EJB, security, transactions, naming, and database access.

This is a *thick* client: The user must install a Java EE Client container runtime environment on the system before executing the client program. Note that this can be hidden from the user by setting up a Web page that invokes an applet that downloads or installs the Java EE Client container; however, the end result is the same: The Java EE Client container is installed on the client system.

## DMZ (not shown)

Many Java EE environments also insert a layer between the client tier and the presentation tier consisting of a traditional HTTP server that serves static content (HTML, images, and media).

Some call this the Web tier, but we do not like using that name because others use the name Web tier interchangeably with the presentation tier. We recommend calling this the de-militarized zone (DMZ) tier.

The DMZ is typically separated from the overall Internet through one firewall that typically lets port 80 traffic through. It is separated from the rest of the infrastructure through another firewall that typically lets through traffic only from

the HTTP server to specific back-end servers. There are two motivations for having an HTTP server separate from the Java EE containers:

- ▶ **Security:** The DMZ, by virtue of the firewall configuration, is more susceptible to attack. If this system is compromised, the content can be wiped and the system reloaded with no loss of data (since all the content is static).
- ▶ **Performance:** HTTP servers are generally optimized to serve static content as fast and as efficiently as possible, and in most situations will do this much faster than a Java EE application server. Therefore, for large environments, implementing separate HTTP servers can lead to a smaller overall Java EE infrastructure.

## Presentation tier

The presentation tier processes input from the client side tier, calls components in the business logic tier, and then sends a response back to the client side tier.

The Java EE Web container supports the execution of the servlet and JSP programming artifacts, manages the HTTP requests, and provides common services (session state, security, transactions, naming, and database access):

- ▶ Servlets provide *100% dynamic content*. They are Java classes that extend the `HttpServlet`, providing specific methods that support interaction with standard HTTP methods (get, post) and request/response objects.

The user selects a submit action on a Web page that points to the Web address for the servlet, and the servlet is executed. The servlet can return the content directly to the browser by printing HTML statements using `out.println` statements or redirect to an HTML or JSP page.

- ▶ JavaServer Pages (JSPs) provide *partially dynamic content*. They are a combination of HTML and Java, and when invoked are compiled by the Web container and run—effectively as servlets. Therefore, the Web container runtime environment actually supports a single run-time programming model: servlets.

The advantage of JSPs are that development tools can provide *what you see is what you get* (WYSIWYG) support. This provides a much more effective development process than the series of `out.println` statements that you code in a servlet.

- ▶ HTML and other static content can also be served up by the Web container. You can choose to do this in small environments (where you do not want to set up an external HTTP server in addition to the Java EE application server) or when you want to control access to the resources (using Java EE declarative security).

Web containers typically contain a minimal HTTP server that directly receives requests from the Web browsers. Therefore, the Java EE application server can be used either with or without an external HTTP server.

### ***Servlets versus JSP***

When should you use a servlet versus a JSP? Typical Java EE best practices suggest considering the balance between HTML and Java:

- ▶ If you are writing a servlet and find that you are creating many `out.println` statements for HTML, you probably should be using a JSP. (Some best practices go even further: If you are using any `out.println` statements, you should switch to using a JSP, or at least split out the `out.println` statements into a separate JSP file).
- ▶ If you are writing a JSP and find that you are including quite a bit of Java statements and are interacting with attributes of the `HttpServlet` interface, you probably should split that code out into a separate servlet.

### **Business logic tier**

The business logic tier provides a framework for executing business logic and for accessing business data in a distributed transactional environment.

The Java EE EJB container provides the services needed to support the access to and execution of business logic. Business logic and data can be accessed concurrently by many different programs on many different systems. The Java EE EJB container provides the underlying transactional services and local/remote transparency needed to support the access to and execution of the business logic:

- ▶ Enterprise JavaBeans (EJBs) provide the Java EE implementation of the business logic, whether data (entities) or logic (session EJBs) or messaging interactions (message-driven beans).

In a non-Java EE environment, management of the transactional interactions and local/remote invocation routines can represent a large percentage of the user code. In a Java EE environment, the business logic tier manages this, dramatically reducing the amount of user written code.

### **Data access in Java EE versus J2EE:**

In earlier versions of Java EE, back-end data was managed through the entity EJB artifact, which provided encapsulated and transactional access to a record-oriented data element with a primary key, such as a row in a relational database. Entity EJBs could be accessed locally by a direct program call within a JVM or remotely by remote method invocation (RMI) between JVMs.

Data was thus rooted in the business logic tier, and all access to data required an EJB element.

In Java EE, the entity EJB and the associated EJB container capabilities are replaced by entities (also called entity objects) and the Java Persistence API (JPA). Entity objects provide similar capabilities as entity EJBs, but they can also be accessed from the Web container and Application Client container.

The entity objects are coordinated by an architectural component called the entity manager, represented by the blue box to the right of the EJB container in Figure 1-1 on page 14.

Standard architectural practices still suggest that data access be maintained in the business logic tier (through session EJBs called facades), but developers now have the flexibility to directly access data from other tiers when appropriate.

Chapter 5, “Create the JPA entities” on page 123 provides detailed information about entities and JPA.

### **Back end tier**

The back end tier represents existing business systems, whether databases or other software systems:

- ▶ The Java Connector Architecture (JCA) (not shown before) provides a common programmatic way of connecting with non-Java EE programs (SAP, Oracle, WebSphere MQ, IBM CICS®) with security, transactions, and performance attributes such as connection pooling.
- ▶ JDBC provides a specific JCA implementation for accessing databases. Earlier JDBC implementations were not implemented on JCA.

## 1.3 Introduction to the scenario

Consider this simple question: *“I am trying to write an application that accesses two back-end databases. How should I access them?”* Your current knowledge of Java EE could lead you to ask questions such as these:

- ▶ *Should I access the two back-end databases using database/JDBC?*
- ▶ *Should I access them using messaging/JMS?*
- ▶ *Should I access them using a Web service?*
- ▶ *Should I access them using an application connection such as JCA?*
- ▶ *Should I utilize data federation or business logic integration (using multiple EJBs)?*
- ▶ *Should I use stored procedures?*
- ▶ *Should I use XML?*
- ▶ *How do I manage the security context of this access?*
- ▶ *How do I manage the transactional integrity?*

The quandary: How do you effectively combine the appropriate Java EE technologies in a manner that provides a responsive, scalable, robust, and flexible solution? How do you learn how to do this in a reasonable period using a document of reasonable length? How do you *experience* Java EE?

This book helps you answer some of these questions by implementing the Donate sample application scenario:

- ▶ *The requirement is to integrate an existing Vacation database (containing an EMPLOYEE table with employee name, number, salary, and available vacation) into a new Donation application that allows employees to donate some of their vacation into a fund that can be used by other employees that need additional time off to deal with family emergencies.*
- ▶ *In the first phase this application should be available through a Web interface and a thick Java client, with security constraints to ensure that only authorized users use the application, and that information such as salary is restricted to management. In subsequent phases this application should be available over Web services and JMS (messaging).*

## Donation sample application

Figure 1-2 represents the complete application, including the Web front end, the thick client, Web services, and messaging (JMS).

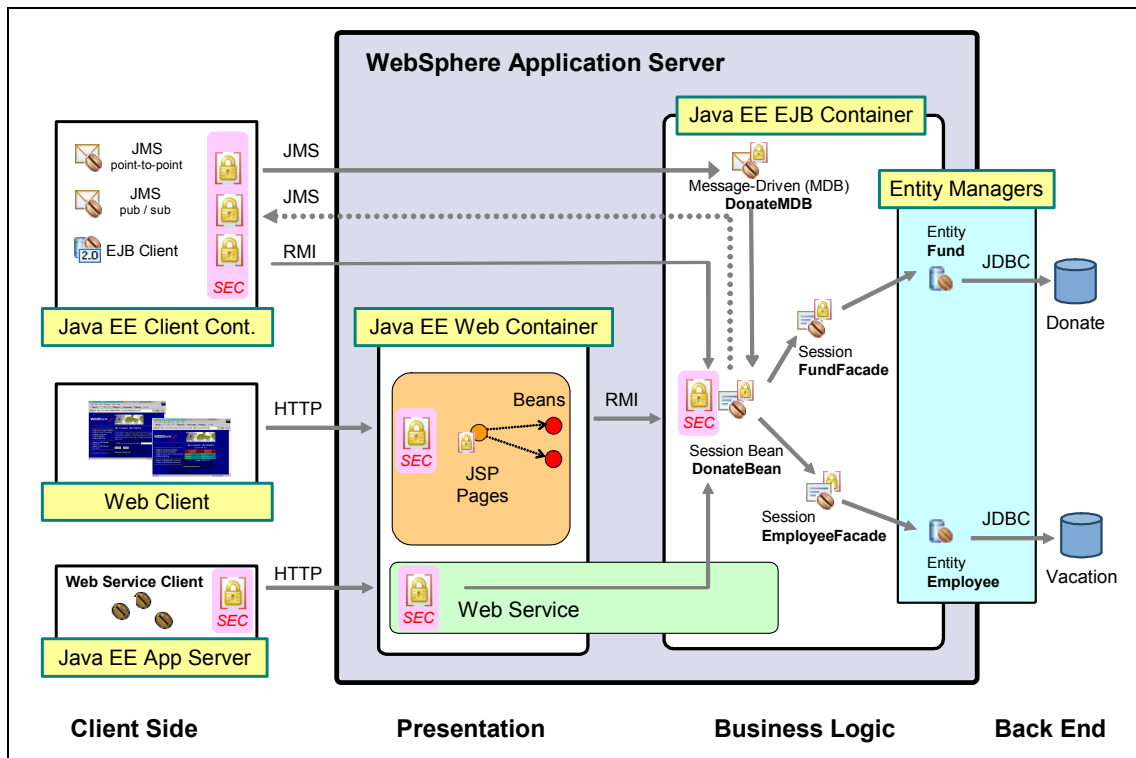


Figure 1-2 Donation sample application scenario

In this book, we implement the application by following the instructions in these parts and chapters:

Part 1, “Preliminary activities” on page 1: Install the required IBM software products and create the initial environment needed to support the application:

- ▶ Chapter 2, “Install and configure software” on page 25: Ensure that the prerequisite products are installed.
- ▶ Chapter 3, “Configure the development environment” on page 53: Prepare the development workspace and test server.
- ▶ Chapter 4, “Prepare the legacy application” on page 85: Extract the application samples needed for this book, and create and populate the Vacation database that represents the current application.



Part 2, “Core Java EE application” on page 121: Create the core Java EE artifacts (data access elements, application business logic elements, Web front end, and thick client):

- ▶ Chapter 5, “Create the JPA entities” on page 123: Create the Employee entity EJB (based on the Employee table in the existing Vacation database).
- ▶ Chapter 6, “Create the entity facade session beans” on page 201: Create the Funds entity EJB and the Funds table/database (based on the Funds entity EJB).
- ▶ Chapter 7, “Create the Donate session bean for the business logic” on page 261: Create the Donate session EJB to transfer vacation from an Employee entity EJB to a Funds entity EJB.
- ▶ Chapter 8, “Create the Web front end” on page 277: Create a Web application that provides access first to the EmployeeFacade session EJB, and then to the Donate session EJB.
- ▶ Chapter 9, “Create the application client” on page 345: Create a thick Java client that invokes the Donate session EJB.
- ▶ Chapter 10, “Implement core security” on page 361: Add the needed users to the user registry (in a file system based registry), configure the test server for Java EE application security, and implement both declarative security (Web and EJB, to restrict access to the overall Web page and EJB) and programmatic security (Web and EJB, to restrict access to the salary field).

Part 3, “Web services” on page 407: Extend the core Java EE application to support Web services:

- ▶ Chapter 11, “Create the Web service” on page 409: Use the tooling wizards to expose the Donate session EJB as a Web service, and create a Web service client to access this Web service.
- ▶ Chapter 12, “Implement security for the Web service” on page 447: Update the Web service and the Web service client to support Web services security using user ID/password for authentication.

Part 4, “Messaging” on page 475: Extend the core Java EE application to support messaging (JMS):

- ▶ Chapter 13, “Create the message-driven bean” on page 477: Create a message-driven bean to receive messages and to call the Donate session EJB. Create a JMS client to send a message to the message-driven bean.
- ▶ Chapter 14, “Add publication of results” on page 517: Update the Donate session EJB to publish the donation request results to a topic representing the employee number. Create a JMS client to subscribe to updates for an employee.

- ▶ Chapter 15, “Implement security for messaging” on page 535: Update the messaging artifacts to support security.

Part 5, “Advanced Java EE 5 and Web 2.0” on page 545: Describes advanced Java EE and non-Java EE capabilities and technologies that can be used to extend the core application:

- ▶ Chapter 16, “Develop a Web 2.0 client” on page 547: Describes how to develop a Web application using Web 2.0 technologies, such as Asynchronous JavaScript and XML (Ajax), Java Script Object Notation (JSON), Representational State Transfer (REST), and the Dojo toolkit.

## 1.4 Jump start

Jump start means *to start or restart rapidly or forcefully*

from Merriam-Webster on-line dictionary, available at this Web address:  
<http://www.merriam-webster.com>

The intention is for most users to implement the Experience sections of this book in a sequential manner: Starting with chapters in Part 1, “Preliminary activities” on page 1 and following in order.



However, readers who are already familiar with parts of Java EE might want to skip chapters or access in a random manner. Or, they might have problems in completing a specific chapter, but would like to continue with rest of the book.

For those readers, this book provides a “jump start” appendix that contains instructions to quickly configure Rational Application Developer and the embedded WebSphere Application Server test environment for each chapter:

- ▶ All readers have to complete the initial setup chapters in Part 1, “Preliminary activities” on page 1:
  - Chapter 2, “Install and configure software” on page 25
  - Chapter 3, “Configure the development environment” on page 53
  - Chapter 4, “Prepare the legacy application” on page 85
- ▶ Readers then follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for a chapter.

## 1.5 Linux variations

This book is intended for use with both Windows and Linux, and where needed contains alternate instructions when there will be significant differences in commands and usage on these two operating systems.

- ▶  Indicates information unique for Windows
- ▶  Indicates information unique for Linux

The most significant differences are in Chapter 2, “Install and configure software” on page 25.

However, this book would be significantly longer and more complex and confusing if it attempted to point out every minor difference between Windows and Linux.

Therefore, to minimize the added complexity, the Experience sections assume that the reader is using Windows, and the sections are written using the file paths and operating system interactions that exist on Windows.

The most obvious differences between the two operating systems are the base directories used in this book (Table 1-4).

*Table 1-4 Directories on Windows and Linux*

Path	Windows	Linux
Rational Application Developer (core product)	C:\IBM\SDP	/opt/IBM/SDP
Rational Application Developer (embedded WebSphere Application Server)	C:\IBM\SDP\runtimes\base_v7	/opt/IBM/SDP/runtimes/base_v7
Samples directory	C:\7827code	\$HOME/7827code
Workspace	C:\workspaces\ExperienceJEE	\$HOME/workspaces/ExperienceJEE

\$HOME is the home directory for the Linux user. For example, if the Linux user is testuser, the home directory is usually /home/testuser. Note that the home directory for root is usually /root.

Other differences include these:

- ▶ The Linux operating system command line requires the exact file name and normally does not include the current directory on the active search path.
- ▶ When browsing for files using the built-in search pop-ups, the selection button in Windows is **Open** and the selection button in Linux is **OK**.



# Install and configure software

In this chapter, we describe the basic steps to install the software used in this book:

- Rational Application Developer 7.5 with the WebSphere Application Server Version 7.0 Test Environment

## 2.1 Required software

The following software package is used in this book:

- ▶ Rational Application Developer for WebSphere Software V7.5 with the embedded WebSphere Application Server V7.0 test environment.

You can skip this chapter if you already have these products installed. However, this book assumes that you installed the product with the directories shown in Table 2-1.

*Table 2-1 Products expected to be installed for this book*

Product	Installation directory used in this book
Rational Application Developer for V7.5 (base product)	<ul style="list-style-type: none"><li>▶ Windows: C:\IBM\SPD</li><li>▶ Linux:/opt/IBM/SDP</li></ul>
Rational Application Developer for V7.5 (embedded WebSphere Application Server V7.0 test environment)	<ul style="list-style-type: none"><li>▶ Windows: C:\IBM\SPD\runtimes\base_v7</li><li>▶ Linux: /opt/IBM/SDP/runtimes/base_v7</li></ul>

If you installed the product features to other locations, remember to adjust the various instructions to reflect the differences.

Detailed information about the requirements and supported operating systems for these products can be found at:

- ▶ Rational Application Developer for WebSphere Software system requirements:  
<http://www.ibm.com/software/awdtools/developer/application/sysreq/index.html>
- ▶ WebSphere Application Server system requirements:  
<http://www.ibm.com/software/webservers/appserv/was/requirements/>

## 2.2 Hardware requirements

The Rational Application Developer support site describes the hardware and software requirements:

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

The documented minimum hardware requirements are:

- ▶ Intel® Pentium® III 800 MHz or higher
- ▶ 1 GB RAM minimum, 2 GB RAM works well
- ▶ 1024 x 768 video resolution or higher
- ▶ 3.5 GB free hard disk space
- ▶ 2 GB temp space during install

The actual disk space and memory for the overall environment is dependent on many factors including the number of feature packs and refresh packs, the number of profiles, and the types of applications that will be developed and deployed. For example, the actual Rational Application Developer and WebSphere Application Server maximum disk space usage observed during the development of this book under both Microsoft Windows and Linux was approximately 10.5 GB. The disk space used by specific components is shown in Table 2-2.

*Table 2-2 Disk space required per component*

Components	Directory (Windows)	Disk space
IBM Installation Manager	C:\IBM\InstallationManager	0.1 GB
Shared Directory	C:\IBM\SDPShared	3.4 GB
Product Directories	C:\IBM\SDP	3.8GB
Workspace	C:\Workspaces\ExperienceJEE	0.1GB
<b>Total</b>		7.4 GB

## 2.3 Software requirements

Rational Application Server 7.5 and WebSphere Application Server are supported on a variety of operating systems, including Microsoft Windows (Visa, XP), Linux distributions (Red Hat, SUSE), AIX, Sun Solaris.

This book was developed and validated on Microsoft Windows XP Professional (Service Pack 2).

Other requirements beyond the base operating system are as follows:

- ▶ Your system must be able to extract ZIP files.
- ▶ Your system must have an Internet browser (such as Microsoft Internet Explorer 6 SP1 or later or Firefox 2 or later).

## 2.4 Install the software (fastpath)

This section describes using the Rational Application Developer V7.5 trial code mechanism to install the software necessary for this book at the latest update level:

- ▶ Rational Application Developer Version 7.5.4 (base version plus Refresh Pack 4) which provides an Eclipse-based integrated development environment.
- ▶ IBM WebSphere Application Server Version 7.0 Test Environment (base version plus Fix Pack 5) which provides a bundled WebSphere Application Server V7.0 runtime.

Trial code is the simplest and most direct mechanism for installing a Rational Application Developer environment. The trial code is installed with a 60 day license, and the code is disabled after that period. However, you can upgrade your trial code to a formal license version as described in 2.5.5, “Rational Application Developer License” on page 49, provided that you have purchased the appropriate license.

This mechanism requires a system with at least 9.5GB of free space for the installation process to succeed:

- ▶ 6.2 GB for the installed software
- ▶ 3.3 GB for the downloaded installation and update files (which are then erased after the installation completes)

The userid used during the installation must be an administrative user. If the installation userid does not have administrative privileges, you can still install the trial code but you must manually download the core Rational Application Developer trial code images and run an alternate installation program (userinst). Refer to 2.5, “Install the software (advanced)” on page 37 for further details.

Products that are installed via the IBM Installation Manager can be installed at the latest service level in one step, with both the base product and the latest service installed together.

The steps described in this section do require an Internet connection both for your browser and for the Java installation programs that are downloaded and installed on your system. If your firewalls blocks this traffic, you will instead need to manually download and install the base software and fixes as described in 2.5, “Install the software (advanced)” on page 37.

While writing this Redbooks publication, we installed IBM Rational Application Developer v7.5 via two methods: over the Internet from the trial download site (as described in this section), and also from a downloaded electronic image on the



local workstation. Both techniques resulted in identical installations, with the exception of the product license.

1. From your browser, open the Rational Application Developer trial page, available at this address:

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

- a. Ensure that the **Try** tab is selected (Figure 2-1).

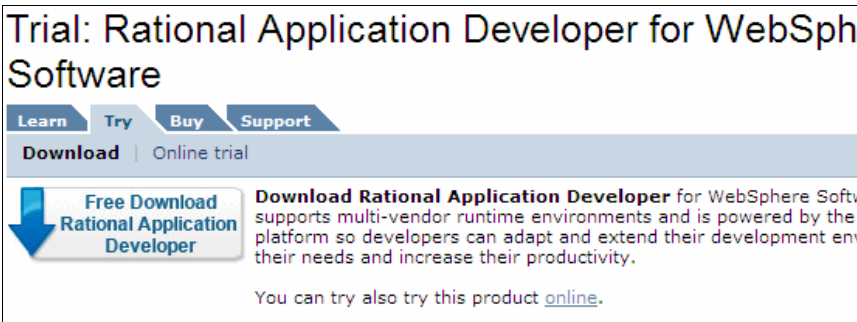


Figure 2-1 Try tab for trial

- b. Select **Web install using IBM Installation Manager (recommended)** (Figure 2-2).

Operating system	Version	Size	Download
<a href="#">Web install using IBM Installation Manager (recommended)</a> Windows, Linux	V7.5	1830MB	
<a href="#">Local install using Download Director or HTTP</a> Windows, Linux	V7.5	2932MB (total)	HTTP

Not sure which method to choose? Check out the [Installation Manager](#) and [Download Director](#) FAQs.

Figure 2-2 Select operating system

2. At the IBM Installation Manager browser page, enter your IBM ID and password and select **Sign in** (Figure 2-3 on page 30).

# IBM Installation Manager

IBM Installation Manager for Software Development Platform

### Returning visitors

IBM ID: (usually e-mail address)\*

→ [Forgot your IBM ID?](#)

→ [Get an IBM ID](#)

Password\*

→ [Forgot your password?](#)

Sign in

### Not registered?

If you do not have a universal IBM user ID, please [register here](#), then return to sign in for this offering.

To find out more about the benefits of having an IBM Registration ID, visit the [IBM ID Help and FAQ](#).

Figure 2-3 Installation Manager sign in

### Behind the scenes:

Certain portions of the IBM.COM Web site such as trial code require users to login with an userid and password. Refer to the Help and FAQ page for further details:

<http://www.ibm.com/account/profile/us?page=regfaqhelp>

3. At the IBM Installation Manager/License browser page, read the license agreements, enable the **I agree** option and select **I confirm**,
4. At the IBM Installation Manager /Download page (Figure 2-4 on page 31):
  - a. Ensure that the **Download using http** tab is selected (the selected tab will be in white).
  - b. Next to Windows or Linux select **Download now**.

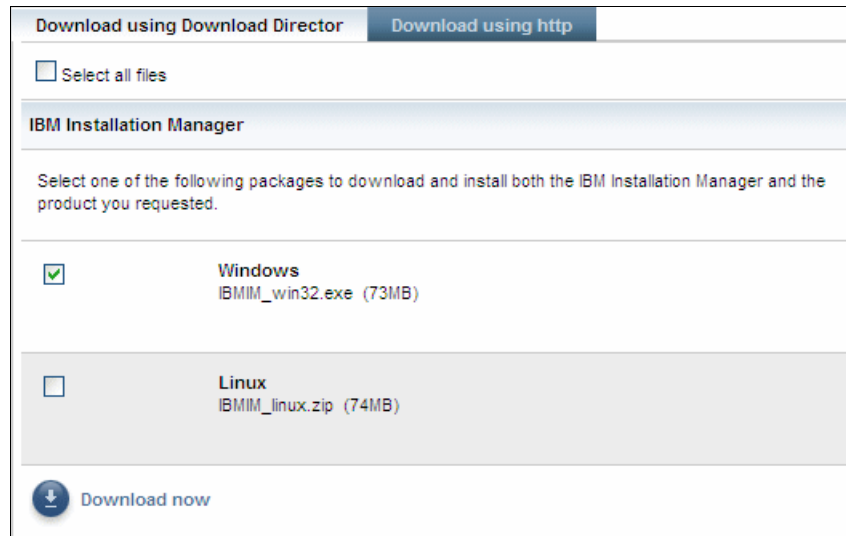


Figure 2-4 Downloading using Installation Manager

### Behind the scenes:

The IBM Installation Manager install program is relatively small and can easily be downloaded using a standard browser download capability. Most users will use the http download at this point.

However, other downloads from the IBM.COM site may involve large files (some exceeding one gigabyte) that may not download reliably over slower internet connection. For those situations, IBM.COM provides the Download Director. From the FAQ page cited below:

Download Director is a Java applet that provides enhanced download capability to IBM customers who download our products from the internet. It provides high-performance, reliable downloads with pause and resume capability. It starts in a separate browser window and provides a graphical user interface (GUI) to specify the download location, and view download status

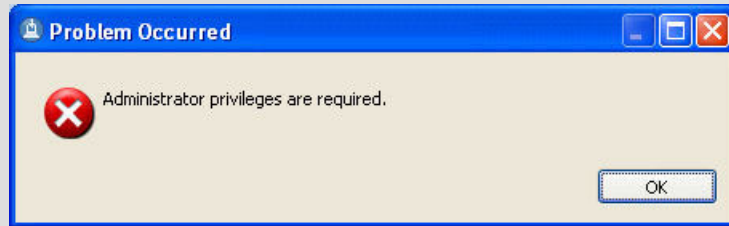
Refer to the Download Director - Frequently Asked Questions Help and FAQ page for further details:

[http://www6.software.ibm.com/dldirector/doc/DDfaq\\_en.html](http://www6.software.ibm.com/dldirector/doc/DDfaq_en.html)

5. After the download completes, execute the file.

### Off Course:

If the installation fails with the following screen you are running as a non-administrative user.



The default installation steps require that the current user have administrative privileges. However, the IBM Installation Manager does provide an alternate program that can be used when installing from product images (whether physical media or downloaded images), as described in the Information Center:

[http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.rad.install.doc/topics/t\\_preinst\\_prep\\_setupdisk\\_start.html](http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.rad.install.doc/topics/t_preinst_prep_setupdisk_start.html)

This option cannot be used when installing the trial code directly over the internet. Instead, you need to download the trial media (select the **Local install using Download Director or HTTP** option) and follow the detailed steps described in 2.5, “Install the software (advanced)” on page 37.

6. At the Password Required pop-up, enter the IBM.COM userid and password that you used earlier and click **OK**.
7. At the IBM Installation Manager/Install Packages pop-up
  - a. Click **Check for Other Versions and Extensions**.
    - At the Search Result pop-up, click **OK**.

### Off course?

This specific step requires that the installation manager have access to the Internet. If your firewall blocks access at this point, you need to follow the steps described in 2.5, “Install the software (advanced)” on page 37. In this more detailed steps, you have the option to configure the IBM Installation Manager to use FTP proxies or HTTP proxies (proxy host or SOCKS host).

- b. At the refreshed IBM Installation Manager/Install Packages pop-up (Figure 2-5), ensure that the following packages and update levels are selected:
- IBM Installation Manager Version 1.3.3
  - IBM Rational Application Developer for WebSphere Software Version V7.5.4
  - IBM WebSphere Application Server Version 7.0 Test Environment Version 7.0.0.5
- c. Click **Next**.

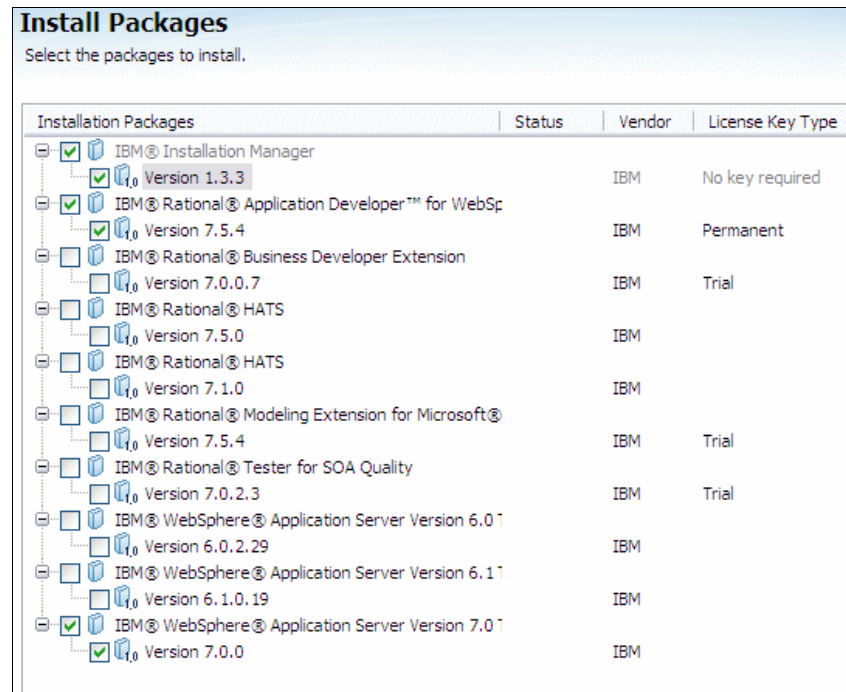


Figure 2-5 Install packages

### Behind the scenes:

The IBM Installation Manager searches specific web pages for online repositories that contain updates for Rational Application Developer and other IBM products that are installed and updated using the Installation Manager.

Repositories can be accessed online (via URLs that associated with each IBM product that is installed and maintained via the Installation Manager) or via local file system paths that refer to the product CDs/DVDs or to downloaded and expanded product images.

Refer to the online IBM Installation Manager documentation for further information about configuring product repositories:

[http://publib.boulder.ibm.com/infocenter/install/v1m0r0/topic/com.ibm.cic.agent.ui.doc/topics/t\\_specifying\\_repository\\_locations.html](http://publib.boulder.ibm.com/infocenter/install/v1m0r0/topic/com.ibm.cic.agent.ui.doc/topics/t_specifying_repository_locations.html)

8. The Progress Information pop-up (Figure 2-6) displays the status of the preparation phase and automatically close when that is completed.

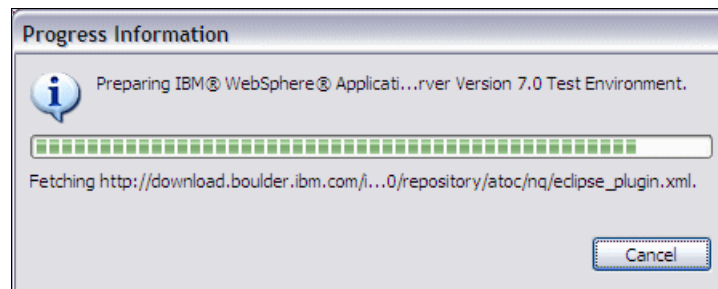


Figure 2-6 Progress information

9. At the Install Packages / Read the following license agreements carefully pop-up, enable **I accept the terms in the license agreements** and click **Next**.
10. At the Install Packages / ... Provide an answer for question 1 pop-up, answer all the questions and then click **Next**.
11. At the Install Packages / Select a location for the shared resources directory and a location for Installation Manager pop-up:
  - a. Set the Shared Resources Directory to: C:\IBM\SPDShared
  - b. Set the Installation Manager Directory to C:\IBM\InstallationManager\eclipse

- c. Click **Next**.

**Behind the scenes:**

The authors prefer to avoid directories with spaces since many programs often fail to properly address the spaces in batch files and command scripts. We are not aware of any specific Rational Application Developer in this area.

Note that the recommended Installation Manager directory has no space between Install and Manager.

- 12. At the Install Packages / A package group ... pop-up:
  - a. Set the Installation Directory to C:\IBM\SDP
  - b. Click **Next**.
- 13. At the Install Packages / .... extend an existing version of Eclipse pop-up:
  - a. Do not enable the Extend an existing Eclipse selection.
  - b. Click **Next**.
- 14. At the Install Packages / Select the translation to install pop-up:
  - a. Optionally enable any of the additional languages
  - b. Click **Next**.
- 15. At the Install Packages / Select the feature to install pop-up (Figure 2-7 on page 36):
  - a. Enable **IBM WebSphere Application Server Version 7.0 Test Environment 7.0.0.5 → Feature Pack for Web 2.0**.
  - b. Click **Next**.

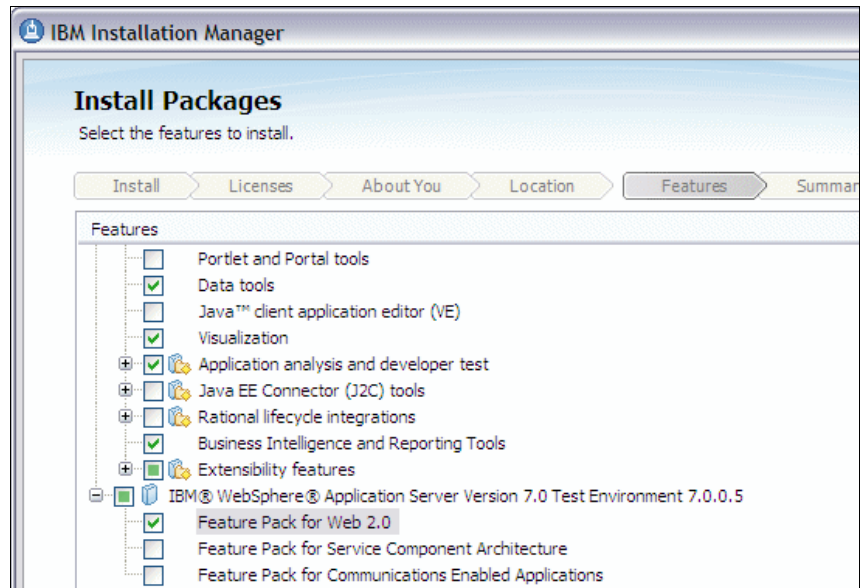


Figure 2-7 Install features

#### Behind the scenes:

This page allows you to alter the specific product features that are installed. The default selections in general are sufficient for this book, but the Feature Pack for Web 2.0 is needed for Chapter 16, “Develop a Web 2.0 client” on page 547

Typically most users accept the default selections. The most common change is to install the Portlet and Portal tools under Rational Application Developer. This feature is not used in this book, but many Rational Application Developer users opt to install it because portlets are commonly used, whether within a portal or a standalone web page.

16. At the Install Packages / Fill in the configurations for packages
  - a. For the Help System configuration, accept the default value of Access Help from the Web
  - b. Click **Next**.
17. At the Install Packages / Fill in the configurations for packages
  - a. For the WebSphere Application Server Version 7.0 Test Environment 7.0.0.5 configuration, de-select the Create a Profile option.



- b. Click **Next**. The Progress Information pop-up will display the status of the preparation phase and automatically close when that is completed.

**Behind the scenes:**

You will manually create the profile in Chapter 3, “Configure the development environment” on page 53.

18. At the Install Packages/Review the summary information pop-up (Figure 2-8), click **Install**.

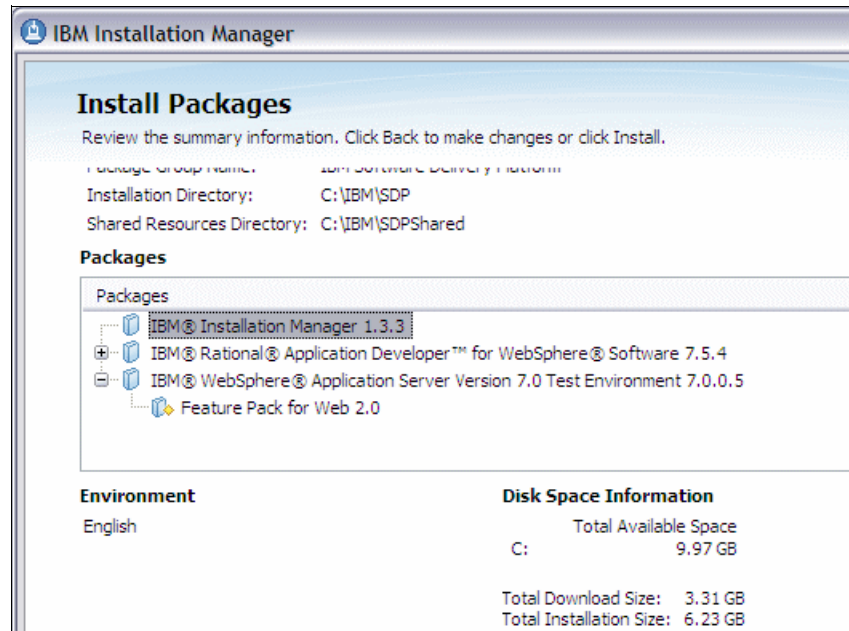


Figure 2-8 Summary information pop-up

19. At the final Install Package pop-up confirming the successful installation, wizard, deselect the option to start IBM Rational Application Developer on exit, and click **Finish** to close the Install Package wizard.

## 2.5 Install the software (advanced)

The IBM Installation Manager 2.4, “Install the software (fastpath)” on page 28 described one sequence for installing Rational Application Developer:

- ▶ The initial IBM Installation Manager version is installed from a file downloaded from the IBM Rational Application Developer trial code site.
- ▶ The IBM Installation Manager installs updates to itself and the overall Rational Application Developer (based and updates) from connects to electronic repositories on the IBM Rational Application trial code site.
- ▶ The product license is left configured with a 60 day trial license.

However, this sequence may not be appropriate for all users, such as those without high speed internet connections or those that purchased formal Rational Application Developer product media.

The Rational Application Developer installation documentation describes the complete set of installation options and techniques.

[http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.rad.install.doc/topics/c\\_intro\\_product.html](http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.rad.install.doc/topics/c_intro_product.html)

The overall installation of Rational Application Develop v7.5 consists of the following major categories:

- ▶ (Optional) Rational Application Developer Product Installation Launchpad
- ▶ IBM Installation Manager installation (base product and updates)
- ▶ Rational Application Developer installation (based product and updates)
- ▶ Post-install Rational Application Developer updates
- ▶ License configuration

The following sections describes some of the other tools and techniques that can be used to install, update, and license Rational Application Developer.

## 2.5.1 Launchpad

The Rational Application Developer initial installation program is called Launchpad. It provides the starting point for installing the core Rational Application Developer product, optional components, and viewing product installation documentation.

From a core product installation perspective, the Launchpad provides the user with the option to install the IBM Installation Manager (see the next section) with the basic configuration needed to locate the Rational Application Developer installation files in a location called a repository.

This Launchpad program is not used when installing Trial code over the internet, and its role/function is replaced by a specially packaged installation manager that is configured to point to the Rational Application Developer repository on the internet.

Launchpad is available from any locations that contain the Rational Application Developer product media:

- ▶ Formal product installation CDs/DVDs
- ▶ Electronic image downloaded from IBM Passport Advantage®
- ▶ Electronic image downloaded from the Trial Code site (for customers that have slow internet connections or need to install on a system that does not have direct internet access)

While writing this Redbooks publication, we installed IBM Rational Application Developer v7.5 from a downloaded electronic image on the workstation. The steps are listed as follows:

1. After downloading all the components of Rational Application Developer v7.5, unzip the files into an installation folder.
2. From there start the Launchpad by executing RAD\_SETUP\launchpad.exe.
3. On the first panel, select the language, for example, English, and click **OK**.
4. Click **Install IBM Rational Application Developer for WebSphere Software** (Figure 2-9 on page 40). This starts the installation of the IBM Installation Manager which will be configured against the source repository that contains the remainder of the Rational Application Developer installation files



Figure 2-9 Install Rational Application Developer for WebSphere Software

## 2.5.2 IBM Installation Manager

IBM Installation Manager is used to install Rational Application Developer. IBM Installation Manager is a program that helps you install the Rational desktop product packages on your workstation. It also helps you update, modify, and uninstall this and other packages that you install. A package can be a product, a group of components, or a single component that is designed to be installed by Installation Manager.

Users typically start the IBM Installation Manager from the Launchpad as described in the previous section. However, you can also start the IBM Installation Manager manually by any of the following steps:

- ▶ Executing the IM\_<platform> package that was downloaded from the trial code site (as is done in “2.4 Install the software (fastpath)” on page 28).
- ▶ Executing the RAD\_SETUP\InstallerImage\_<platform>\install(.exe) for users with administrative privileges
- ▶ Executing the RAD\_SETUP\InstallerImage\_<platform>\userinstl(.exe) for users with Administrative privileges

While writing this Redbooks publication, we followed these steps after starting the IBM Installation Manager installation via the Launchpad (as described in 2.5.1, “Launchpad” on page 38).

1. Note the default version of the IBM Installation Manager and click **Next**.

#### **Behind the scenes:**

The exact version of IBM Installation Manager will vary depending on your source media:

- ▶ 1.2 with the initial Rational Application Developer V7.5 media
- ▶ 1.3.1 with the downloaded V7.5.3 trial media
- ▶ 1.3.3 with the over-the-internet V7.5(.4) trial code installation

At this point you may also choose to click **Check for Other Versions and Extension**. If you have an internet connection, this will connect with the IBM Rational Application Developer support site and check for newer versions of the IBM Installation Manager and automatically install both the base and the updates.

2. Accept the license agreement and click **Next**.
3. Set the installation directory to C:\IBM\InstallationManager\eclipse and click **Next**.
4. Click **Install** and, wait for the installation to finish,
  - a. At the successful installation confirmation, click **Restart installation Manager**. The IBM Installation Manager will start.



Figure 2-10 IBM Installation Manager

### Behind the scenes:

There are six wizards in the Installation Manager that manage a software package through its lifecycle.

- ▶ The Install wizard walks you through the installation process.
- ▶ The Update wizard searches for available updates to packages you have installed.
- ▶ With the Modify wizard, you can modify certain elements of a package you have already installed.
- ▶ The Manage Licenses wizard helps you set up the licenses for your packages.
- ▶ Using the Roll Back wizard, you can revert back to a previous version of a package.
- ▶ The Uninstall wizard removes a package from your computer.

You will use the Install wizard in 2.5.3, “Installing IBM Rational Application Developer” on page 43 and the Manage License wizard in 2.5.5, “Rational Application Developer License” on page 49

- b. If your system has firewalls that require special proxy configurations, from the open IBM Installation Manager select **File** → **Preferences**. At the resulting Preferences pop-up:
  - a. On the left side, expand to **Internet** → **FTP Proxy** or **Internet** → **FTP Proxy**.
  - b. Configure the specific settings as required in your network environment. The host and port definitions shown in Figure 2-11 are fictitious, you must use the values that are appropriate for your environment
  - c. Click **OK** to save the preferences and close the pop-up.

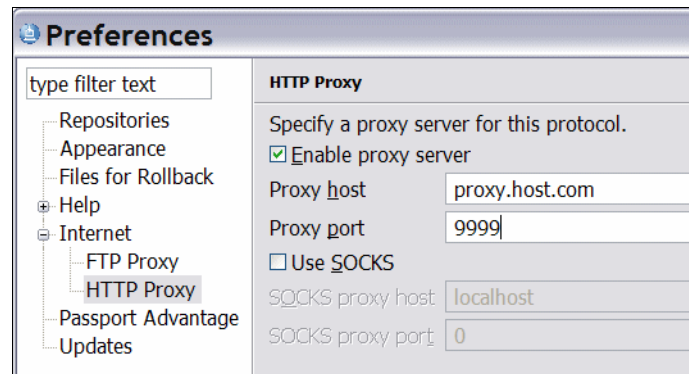


Figure 2-11 HTTP proxy settings

### 2.5.3 Installing IBM Rational Application Developer

After the IBM Installation Manager is installed, it is used to install specific products, in this case Rational Application Developer.

While writing this Redbooks publication, we followed these steps after completing the IBM Installation Manager installation as described in the previous section

1. At the IBM Installation Manager pop-up, click **Install**.

#### Off Course?

If you have internet connectivity and if you did not update the IBM Installation Manager in the previous section, you will receive a prompt at this point indicating that it must be updated before continuing.

- a. Click **Yes** to install the updates,
- b. Click **OK** to restart the IBM Installation Manager
- c. Click **Install** to install Rational Application Developer

2. At the Install Packages/ Select the packages to install pop-up:
  - a. Click **Check for Other Versions and Extensions**.

#### **Off course?**

Internet connectivity issues may prevent this step from working. If you cannot configure the FTP/HTTP proxies as described in the previous section, you will need to manually download the updates and configure Installation Manager to use the local update images.

Refer to 2.5.4, “Rational Application Developer updates” on page 44 for further information.

- b. Select **IBM Rational Application Developer for WebSphere Application Server** and select **IBM WebSphere Application Server Version 7.0 Test Environment**
    - c. Click **Next**.
  3. Continue the installation steps as described at step 8 on page 34. The steps are fundamentally the same from this point forward in the Rational Application Developer installation process.

Leave the IBM Installation Manager pop-up open since this may be used in subsequent sections.

## **2.5.4 Rational Application Developer updates**

Rational Application Developer has three separate components that may require updates:

- ▶ IBM Installation Manager
- ▶ Rational Application Developer core product
- ▶ Rational Application Developer extensions (such as the WebSphere Application Server Version 7.0 Test Environment)

Rational Application Developer updates are typically accessed over the internet and installed concurrently with the base product. However, there are certain situations where this may not be possible:

- ▶ Direct access may not be possible due to firewall issues or if the target system lacks internet access.
- ▶ The update may be new and only became available after the installation



## IBM Installation Manager updates

The default IBM Installation Manager V1.2 must be updated before applying later Rational Application Developer updates. For example, the V7.5.4 refresh pack will fail with the error message (Example 2-1) when using the default installation manager:

### *Example 2-1 Refresh pack failure error message*

---

The "IBM Rational Application Developer for WebSphere Software 7.5.4" installation package requires an Installation Manager with a version tolerance of "1.3.0" (internal version tolerance "1.3.0").  
The running Installation Manager version is "1.2.0" (internal version "1.2.0.20080918\_1021")

---

If your system has internet connectivity, the newer IBM Installation Manager updates will be automatically detected and installed whenever you start the IBM Installation Manager.

For systems that lack internet connectivity or that have firewall issues, you must manually download and apply the update. 1.3.2 and the updates are available at this Web address:

<http://www.ibm.com/support/docview.wss?uid=swg24024055>

As taken from this web page, *IBM Installation Manager, Version 1.3.2*, the page describes how to apply the update:

1. Download the com.ibm.cic.agent.offering.zip fix pack from the FTP or Download Director (DD) site found in the Download package table below.

**Note:** Do not download a platform specific zip file like agent.installer.win32.win32.x86.zip.

2. Extract the compressed file in an appropriate directory. For example, extract to C:\temp.
3. Add the fix pack repository location in IBM Installation Manager:
  - a. Start IBM Installation Manager.
  - b. On the Start page of Installation Manager, click **File > Preferences**, and then click **Repositories**.
  - c. The Repositories page opens.
  - d. On the Repositories page, click **Add Repository**.
  - e. In the Add repository window, browse to or enter the file path to the repository.config file, which is located in the directory where you extracted

the compressed files. For example, enter C:\temp\ or C:\temp\repository.config and then click **OK**.

4. On the start page of the Installation Manager, click **Update**.

The IBM Installation Manager will detect that a newer version is available and prompt you to accept the update.

## Rational Application Developer core product updates

The base Rational Application Developer support page is available at this Web page:

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

The Recommend fixes link off this page points to the current fix levels, are available at this address:

<http://www.ibm.com/support/docview.wss?uid=swg27007823>

The current update level Rational Application Developer for WebSphere Software, Version 7.5.4, found at this address:

<http://www.ibm.com/support/docview.wss?uid=swg24024367>

The Rational Application Developer for WebSphere Software, Version 7.5.4 download page describes how to configure the Installation Manager to access the downloaded images, this instructions are as follows:

1. Download the fix pack files listed in the download table found on the Web page.
2. Extract the compressed files in an appropriate directory. For example, choose to extract to C:\temp\update
3. Add the fix pack's repository location in IBM Installation Manager:
  - a. Start IBM Installation Manager.
  - b. On the Start page of Installation Manager, click **File > Preferences**, and then click **Repositories**. The Repositories page opens.
  - c. On the Repositories page, click **Add Repository**.
  - d. In the Add repository window, browse to or enter the file path to the diskTag.inf file, which is located in the disk1 sub-directory in the directory where you extracted the compressed files and then click **OK**. For example, enter C:\temp\updates\disk1\diskTag.inf.
  - e. Click **OK** to close the Preference page.

If the base product has not been installed, these updates can be installed concurrently with the base product by checking for other versions and extensions.

If the base product has already been installed, the update release notes describe how to apply the updates, these notes are found at this address:

<http://www.ibm.com/support/docview.wss?uid=swg27016544>

The instructions are as follows:

1. Start IBM Installation Manager.
2. On the Start page of Installation Manager, click **Update**.
3. If a new version of Installation Manager is found, you are prompted to confirm that you want to install it before you can continue. Click **OK** to proceed. Installation Manager automatically installs the new version, stops, restarts, and resumes.
4. In the Update packages wizard, select the Installed Location for Rational Application Developer for WebSphere Software and click **Next**. Installation Manager searches for updates in the Rational Application Developer for WebSphere Software repository on the Web, as well as any repository locations that you entered. A progress indicator shows the search is taking place.
5. By default, recommended updates are displayed and selected on the Update Package page. Ensure that you select **Version 7.5.4** for Rational Application Developer for WebSphere Software, and then click **Next**.
6. On the Licenses page, read the license agreements for the selected updates. On the left side of the License page, the list of licenses for the updates you selected is displayed; click each item to display the license agreement text. If you agree to the terms of all the license agreements, click I accept the terms of the license agreements and click **Next**.
7. On the Features page, select the features that you want to update and complete the following steps:
  - a. To learn more about a feature, click the feature and review the brief description under **Details**.
  - b. If you want to see the dependency relationships between features, select **Show Dependencies**. When you click a feature, any features that depend on it and any features that are its dependents are shown in the Dependencies window. As you select or exclude features in the packages, Installation Manager will automatically enforce any dependencies with other features and display updated download size and disk space requirements for the installation.
  - c. Click **Next**.

- d. On the Summary page, review information displayed, and then click **Update**. A progress indicator shows the percentage of the installation completed.
- e. When the update process completes, a message that confirms the success of the process is displayed near the top of the page. Click **View log file** to open the log file for the current session in a new window. You must close the Installation Log window to continue.

## Rational Application Developer extension updates

WebSphere Application Server provides the Update Installer mechanism for applying fix packs to product installations. This mechanism could be used for the embedded test environment within Rational Application Developer, but it would present another tool/mechanism that the Rational Application Developer administrator would need to learn.

Therefore, Rational Application Developer wraps WebSphere Application Server fixpacks into Rational refresh packs that can be applied using the Installation Manager. IBM WebSphere Application Server 7.0 Test Environment Extension Update v7.0.0.5 install instructions are available at this address:

<http://www.ibm.com/support/docview.wss?uid=swg24024307>

The instructions taken from this web page are as follows:

1. Download the update files listed in the download table shown on the Web page.
2. Extract the compressed files in an appropriate directory. For example, choose to extract to C:\temp\update.
3. Add the update's repository location in IBM Installation Manager:
  - a. Start IBM Installation Manager.
  - b. On the Start page of Installation Manager, click **File > Preferences**, and then click **Repositories**. The Repositories page opens.
  - c. On the Repositories page, click **Add Repository**.
  - d. In the Add repository window, browse to or enter the file path to the diskTag.inf file, which is located in the disk1 sub-directory in the directory where you extracted the compressed files and then click **OK**. For example, enter C:\temp\updates\disk1\diskTag.inf.
  - e. Click **OK** to close the Preference page.

The actual installation of the updates will occur automatically when you run the Update Installer update wizard, as described above in “Rational Application Developer core product updates” on page 46.

## 2.5.5 Rational Application Developer License

The Rational Application Developer trial code is installed with a 60 day license, and the product operation will be disabled after that time period. The license status is available via the Manage License wizard in the IBM Installation Manager:

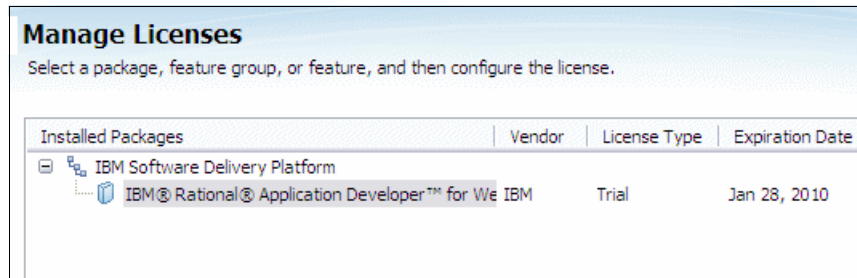


Figure 2-12 Managing licences in the Installation Manager

The IBM Installation Manager is able to upgrade from a trial license to a permanent license without reinstallation of Rational Application Developer, provided you have a valid license activation kit.

IBM Rational offers three types of product licenses (activation kits):

- ▶ Authorized User License
- ▶ Authorized User Fixed Term License (FTL)
- ▶ Floating License

The best choice for your organization depends upon how many people use the product, how often they require access, and how you prefer to purchase the software. The product licenses are:

- ▶ Authorized User License

An IBM Rational Authorized User License permits a single, specific individual to use a Rational software product. Purchasers must obtain an Authorized User License for each individual user who accesses the product in any manner. An Authorized User License cannot be reassigned unless the purchaser replaces the original assignee on a long-term or permanent basis.

- ▶ Authorized User Fixed Term License

An IBM Rational Authorized User Fixed Term License (FTL) permits a single, specific individual to use a Rational software product for a specific length of time (the term). Purchasers must obtain an Authorized User FTL for each individual user who accesses the product in any manner. An Authorized User FTL cannot be reassigned unless the purchaser replaces the original assignee on a long-term or permanent basis.

► Floating License

An IBM Rational Floating License is a license for a single software product that can be shared among multiple team members; however, the total number of concurrent users cannot exceed the number of floating licenses you purchase.

To use floating licenses, you must obtain floating license keys and install them on a Rational License Server. The server responds to end-user requests for access to the license keys; it will grant access to the number of concurrent users that matches the number of licenses the organization purchased.



## Installing the license for Rational Application Developer

You have two options on how to enable licensing for Rational Application Developer:

- Importing a product activation kit
- Enabling Rational Common Licensing to obtain access to floating license keys

In this section, we show you how to import a product activation kit:

1. Start the IBM Installation Manager.
2. At the IBM Installation Manager main page, click **Manage Licenses**.
3. At the Manage Licenses pop-up:
  - a. Select the installed Rational Application Developer package.
  - b. Enable the **Import product Activation Kit** option
  - c. Click **Next**.
4. At the Import Activation Kit pop-up, enter the file name and path into the Repository field and click **Next**.
5. At the Licenses pop-up, select **I accept the terms in the license agreements** and click **Next**.
6. At the Summary pop-up, click **Finish**.
7. At the Import License pop-up, note the message indicating that the license was imported, and click **Finish**.
8. Back at the IBM Installation Manager main page, open the Manage Licenses wizard and verify that the license was successfully update.
  - a. Close the Manage License wizard and IBM Installation Manager when done.

Manage Licenses			
Select a package, feature group, or feature, and then configure the license.			
Installed Packages	Vendor	License Type	Expiration Date
<div> <div>  <div>IBM Software Delivery Platform</div> </div> <div>  <div>IBM® Rational® Application Developer™ for V6 IBM</div> </div> </div>	IBM	Permanent	

*Figure 2-13 The permanent license*







## Configure the development environment

In this chapter, we configure the Rational Application Developer for WebSphere Software development environment to support the Experience Java EE! scenario.

## 3.1 Learn!

The Rational product suite helps businesses and organizations manage the entire software development process. Software modelers, architects, developers, and testers can use the same team-unifying Rational Software Delivery Platform tooling to be more efficient in exchanging assets, following common processes, managing change and requirements, maintaining status, and improving quality.

IBM Rational Application Developer for WebSphere Software v7.5 is an integrated development environment and platform for building Java Platform Standard Edition (Java SE) and Java Platform Enterprise Edition (Java EE) applications with a focus on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal.

The Application Developer is a full suite of development, analysis and test, and deployment tools for rapidly implementing Java SE and EE, Portal, Web and Web 2.0, Web services, and SOA applications. New in version 7.5, it supports Java EE 5.0, EJB 3.0, JPA, and Web 2.0 with Ajax and Dojo development.

Additional Learn resources are available at the following Web sites:

- ▶ Eclipse.org home page:  
<http://www.eclipse.org>
- ▶ Eclipse Web Tools Platform Project:  
<http://www.eclipse.org/webtools>

### 3.1.1 Eclipse Project

The Eclipse Project is an open source software development project devoted to creating a development platform and integrated tooling. Figure 3-1 on page 55 shows the high-level Eclipse Project architecture and shows the relationship of the following sub projects:

- ▶ Eclipse Platform
- ▶ Eclipse Java Development Tools (JDT)
- ▶ Eclipse Plug-in Development Environment (PDE) Figure 3-1 on page 55

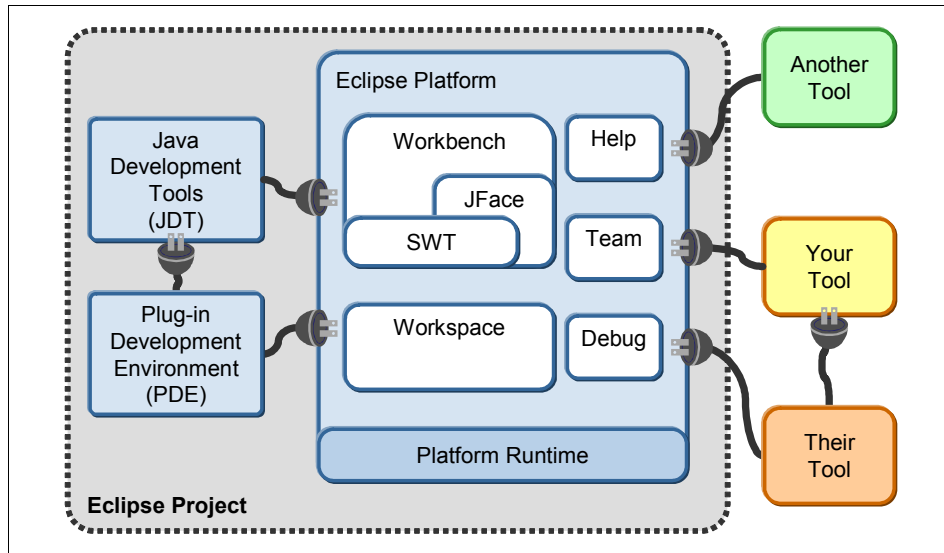


Figure 3-1 Eclipse project overview

With a common public license that provides royalty free source code and world-wide redistribution rights, the Eclipse Platform provides tool developers with great flexibility and control over their software technology.

Industry leaders such as IBM, Borland, Merant, QNX Software Systems, Red Hat, SUSE, TogetherSoft, and WebGain formed the initial eclipse.org board of directors of the Eclipse open source project.

The user sees a single-integrated product and is generally not aware or not concerned about which component a specific feature or function came from.

Both Eclipse and WTP were seeded with code submissions from IBM that were based on actual IBM products:

- ▶ Eclipse was based on the tooling platform initially released with WebSphere Studio Application Developer v4.x.
- ▶ WTP was based on the J2EE tooling in Rational Application Developer v6.x.

### 3.1.2 Workbench and Workspace

Rational Application Developer (and any Eclipse based product) is a graphical development environment that provides a set of capabilities. In this book, the graphical application is called the *workbench*.

The workbench operates against a collection of resources called a *workspace* that provides a contained and isolated environment. Each time the workbench starts, the user is prompted to select a specific workspace.

A workspace typically contains the following components:

- ▶ **Resources**, which represent specific artifacts that the developer will create and test.

A resource can represent a discrete file, such as a Java class or an XML data file, or an abstract construct, such as an EJB or a Web service.

- ▶ **Projects** that allow the grouping of resources.

Projects can be simple, which means they can contain any artifact, or they can be specific to a technology or standard such as an EJB project or an enterprise application project.

- ▶ **Capabilities** to manipulate the various projects and resources.

Capabilities are provided through plug-ins. Products such as the Rational Application Developer workbench come with a built-in set of plug-ins, but you can also download and install plug-ins from other sources as well. The Eclipse Foundation manages the following plug-in download site:

<http://marketplace.eclipse.org/>

- ▶ **Test capabilities** that allow the testing and debugging of projects and resources.

These can include built-in language environments such as Java, standard transformations such as XSL, and specific runtime test environments such as Java EE application servers.

A workspace is implemented as a file system subdirectory structure, but it does allow you to link in artifacts that exist in other locations in the file system.

## 3.2 Extract the base sample code file

The **SG247827\_BASE.ZIP** file contains various directories and files that are used in the scenarios (Table 3-1) and can be downloaded from the following link:



[ftp://www.redbooks.ibm.com/redbooks/SG247827/SG247827\\_BASE.ZIP](ftp://www.redbooks.ibm.com/redbooks/SG247827/SG247827_BASE.ZIP)

Refer to Appendix B, “Additional material” on page 629 for additional download instructions if the above link does not download the file. An additional samples file contains the project interchanges needed for Appendix A, “Jump start” on page 615.

Table 3-1 Sample code content: **SG247827\_BASE.ZIP**

Directory or file	Description
7827code	Master director
../Web2.0	GIF files for “Chapter 16. Develop a Web 2.0 client” on page 547
../snippets experiencejee_snippets.xml	Code snippets used to create various portions of the Java EE application. The snippets are available in text format for copy/paste, and in an XML file that can be imported into the workspace (see 3.7, “Import the sample code snippets” on page 78).

1. Extract the samples file **SG247827\_BASE.ZIP**:



-  **Windows** Unzip to C:\ to create C:\7827code.
-  **Linux** Unzip to \$HOME/ to create \$HOME/7827code.

\$HOME is the home directory for the Linux user. For example, if the Linux user is testuser, the home directory is usually /home/testuser.

## 3.3 Create the workspace



In this section we create a single workspace to use for all scenarios in this book:

1. Start the Rational Application Developer workbench

-  **Windows** Programs menu: **Programs** → **IBM Software Delivery Platform** → **IBM Rational Application Developer 7.5** → **IBM Rational Application Developer**
-  **Linux** Applications → IBM Software Delivery Platform → IBM Rational Application Developer 7.5

2. At the Workspace Launcher pop-up (Figure 3-2 on page 58):

a. Set the Workspace path as follows:

-  **Windows** C:\workspaces\ExperienceJEE
-  **Linux** \$HOME/workspace/ExperienceJEE

b. Do **NOT** select **Use this as the default and do not ask again**.

c. Click **OK**.

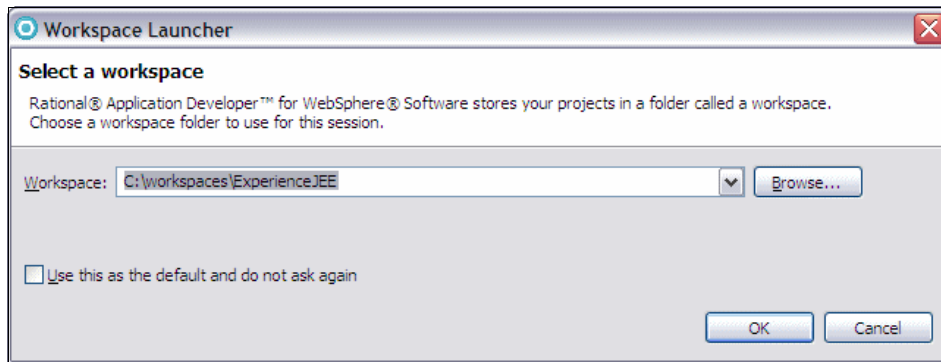


Figure 3-2 Start workspace

### Behind the scenes:

If the specified folder does not exist, the workbench creates it and initializes the workspace by adding a *.metadata* folder, where metadata information will be stored.

You did not select **Use this workspace as the default and do not ask again**, because it would do exactly what it says: It would no longer give you the option to set the workspace directory.

If you accidentally set this, you can re-enable the workspace prompt:

1. In the workbench action bar, select **Window** → **Preferences**.
2. At the Preferences pop-up, select **General** → **Startup and Shutdown**, then select **Prompt for workspace on startup** on the right side, and click **OK**.

3. After the workspace initializes, close the Welcome view by clicking the **X** next to Welcome (Figure 3-3). Note that you can open the Welcome view at any time by selecting **Help** → **Welcome** from the menu bar.

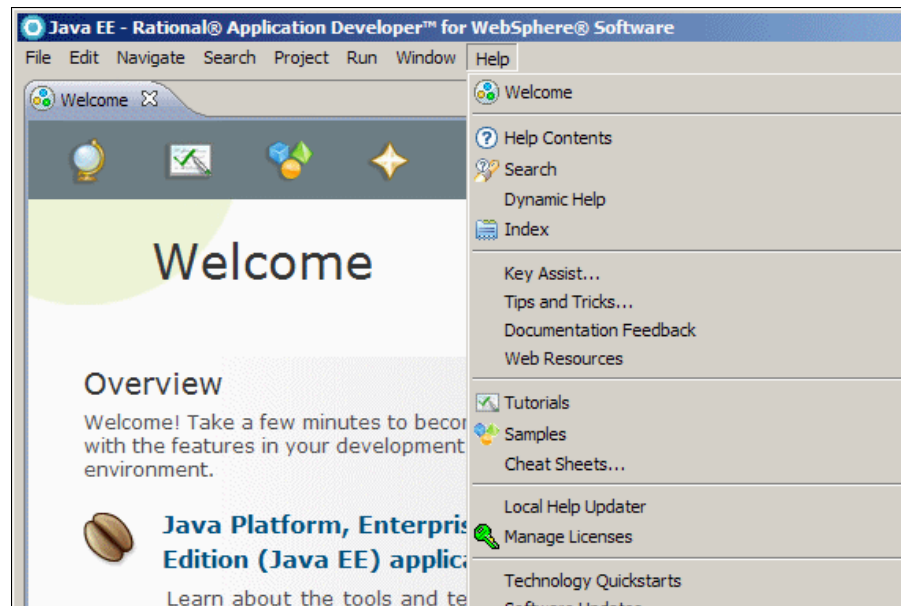


Figure 3-3 Welcome window

4. The workspace title bar in the upper left (Figure 3-4) and the perspective icon in the upper right (Figure 3-4) both indicate the current perspective. With the Ganymede distribution, the default perspective is **Java EE**.

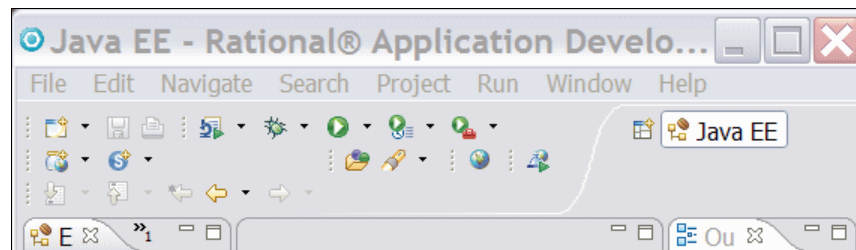


Figure 3-4 Java EE workspace

5. The upper left pane of the workspace consists of views that provide a representation of artifacts in the workspace. The default view for the Java EE perspective is the Enterprise Explorer (Figure 3-5 on page 60).

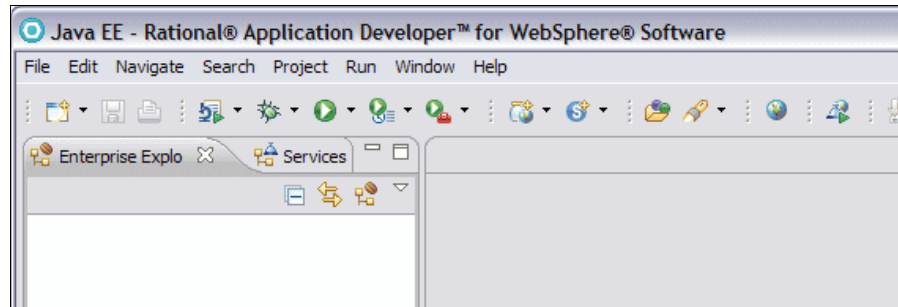


Figure 3-5 Java EE workspace Enterprise Explorer

### Behind the scenes:

Eclipse and eclipse plug-ins provide many different ways of representing the workspace contents. The Java EE perspective shown in Figure 3-5 automatically starts up with two such presentations: Enterprise Explorer and Services.

The Navigator view provides a strict representation of the file based artifacts in the workspace, and you can open this view by selecting by selecting **Window** → **Show View** → **Navigator** from the workspace action bar.

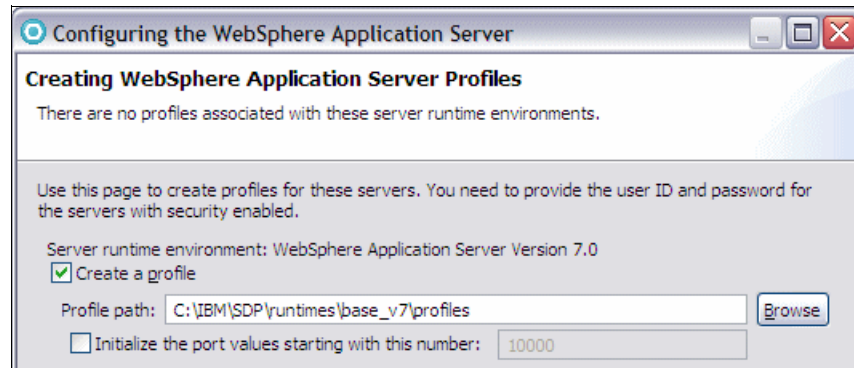
Other views provide an abstract representation that is meaningful in the context of a specific development role. For example, the DataSource Explorer show the fusion of workspace connection definition and information from the actual relational databases to provide a representation that will allow the developer to create artifacts that interact with the database content and structure.

Some of the abstract workspace views and associated perspectives that you will see in this book include:

- ▶ DataSource Explorer (Database Development perspective)
- ▶ Enterprise Explorer (Java EE and Web perspectives)
- ▶ Services (Java EE perspective)
- ▶ Project Explorer (JPA perspective)
- ▶ Package Explorer (Java perspective)

6. If Figure 3-6 on page 61 pop-up appears, close it (click on the red **x** in the upper right). You will create a profile in the next section (3.4, “Create the ExperienceJEE server profile” on page 62).





*Figure 3-6 Configuring the WebSphere Application Server*


### **Behind the scenes:**

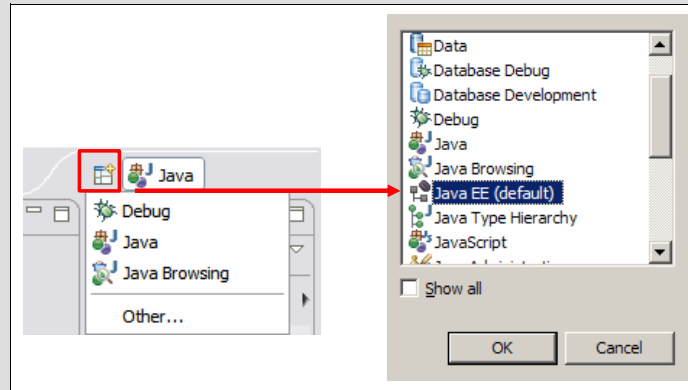
Eclipse-based products contain one or more perspectives that contain a set of default views and capabilities. Each perspective is tailored for a specific development role (in this case Java EE development), only showing views and capabilities that apply for that role.

Typical perspectives used for this document are Java EE, Web, and Java.

### Off course?

If the current perspective is not **Java EE**, you can switch to the perspective through two different options:

1. Click the **Perspective** icon [  ], located at the upper right, and select **Other** from the pull-down.



- a. At the Open Perspective pop-up, select **Java EE (default)** and click **OK**.
2. In the workbench action bar, select **Window** → **Open Perspective** → **Other**, and at the Open Perspective pop-up, select **Java EE**.

## 3.4 Create the ExperienceJEE server profile

The runtime environment configuration for a WebSphere Application Server V7.0 instance is defined in a set of files called *profiles*. These profiles contain the complete run-time definition of a server, including which enterprise applications are installed. WebSphere Application Server typically creates a default profile (normally called AppSrv01), but if you followed the installation instructions in Chapter 2, “Install and configure software” on page 25 there should be no profiles defined at this point.

Each profile also contains a copy of the standard WebSphere Application Server commands (such as **startServer**, **stopServer**, and **wsadmin**) that by default operate against that profile. The commands in the default command directory (C:\IBM\SDP\runtimes\base\_v7\bin) operate against the default profile, unless overridden by the `profileName` parameter.

When working with Rational Application Developer and other Eclipse-based development environments, it is advisable to have a separate profile for each significant application and development environment.

Attempting to use a single server for all workspaces may result in the following:

- ▶ Errors due to conflicts in applications—both being present and being absent.
- ▶ Errors in conflicts in configuration settings, where one workspace requires a configuration setting that conflicts with another workspace.

To create the profile for the new server:

1. In the workbench action bar, select **Window** → **Preferences**.
2. At the open Preferences pop-up, on the left select **Server** → **WebSphere Application Server**.
3. On the right in the WebSphere pane (Figure 3-7), next to the WebSphere profiles defined... list, click **Run Profile Management Tool...** (in the lower right) to start the wizard.

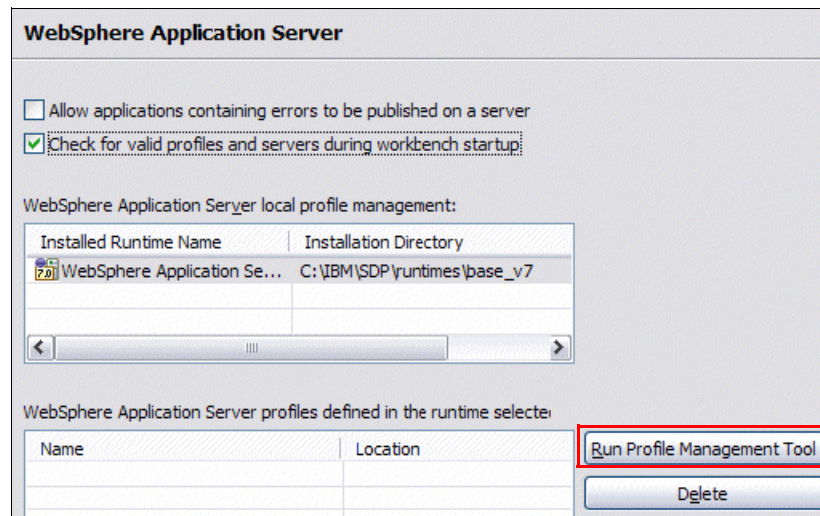


Figure 3-7 WebSphere Application Server pane

### Behind the scenes:

The Create button launches the WebSphere Application Server graphical profile management. This tool can also be invoked directly from the command line:

- ▶ **Windows**  
C:\IBM\SDP\runtimes\base\_v7\bin\ProfileManagement\pmt.bat
- ▶ **Linux**  
/opt/IBM/SDP/runtimes/base\_v7/bin/ProfileManagement/pmt.sh

4. At the Welcome to the Profile Management tool pop-up(Figure 3-8), click **Launch Profile Management Tool.--**

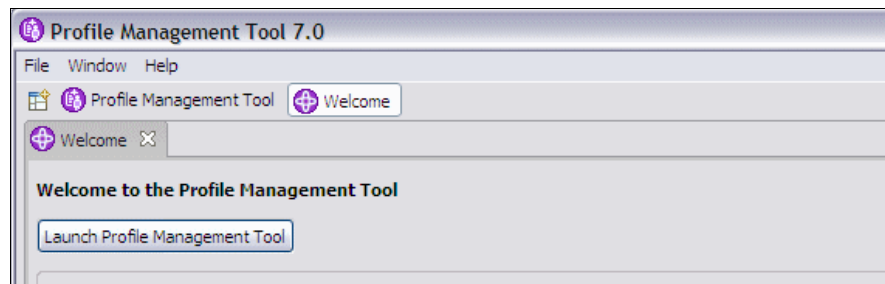


Figure 3-8 Profile Management Tool Welcome

5. In the profile list (Figure 3-9), click on **Create** button on the right.

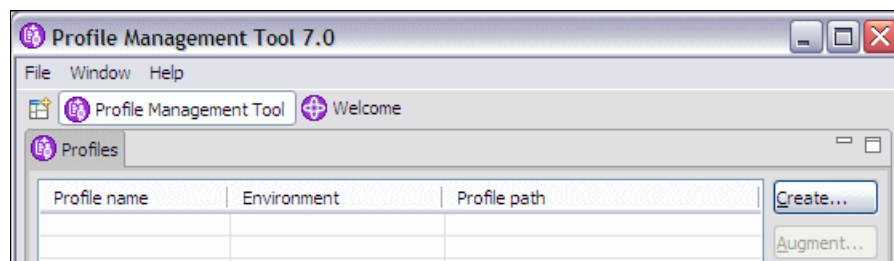


Figure 3-9 Profile Management Tool using create

6. At the Environment Selection pop-up (Figure 3-10 on page 65), ensure that **Application Server** is selected, and click **Next**.

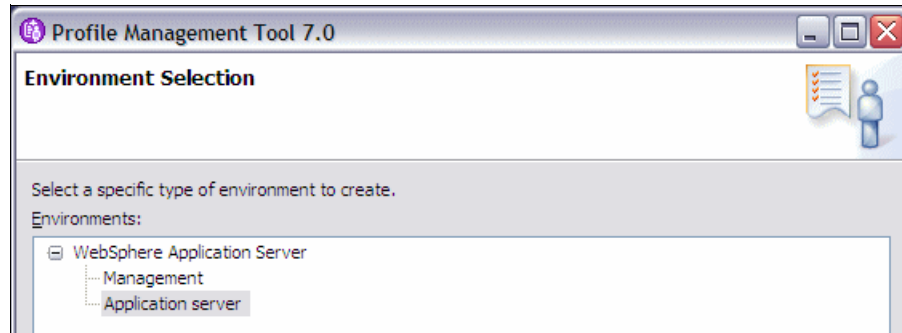


Figure 3-10 Environment Selection popup

7. At the Profile Creation Options pop-up, select **Advanced profile creation**, and click **Next**.
8. At the Optional Application Deployment pop-up, accept the defaults (with both Deploy the administrative console and Deploy the default application selected) and click **Next**.
9. At the **Profile Name and Location** pop-up (Figure 3-11 on page 66):
  - a. Set the Profile Name to **ExperienceJEE**
    - Profile Directory:
      - **Windows**  
C:\IBM\SDP\runtimes\base\_v7\profiles\ExperienceJEE
      - **Linux**  
/opt/IBM/SDP/runtimes/base\_v7/profiles/ExperienceJEE
  - b. Select **Create the server using the development template**.
  - c. Click **Next**.

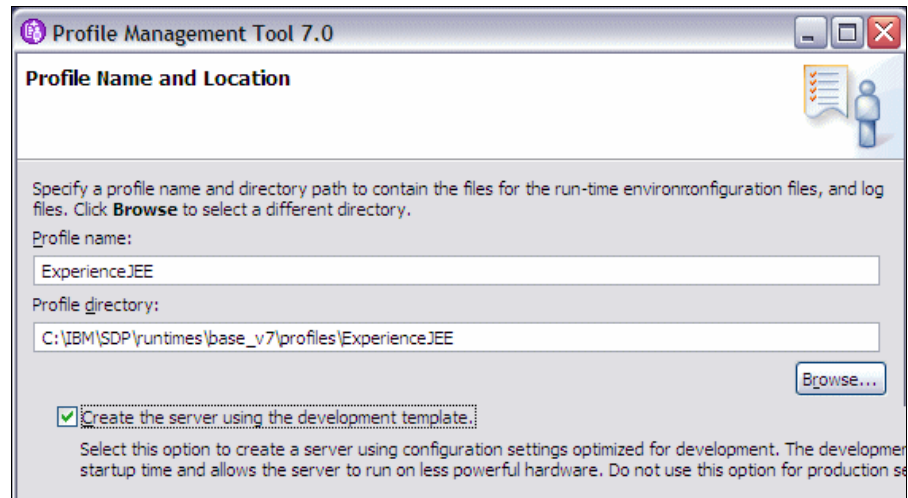


Figure 3-11 Profile Name and Location popup

10. At the Node and Host Names pop-up (Figure 3-12):
  - a. Change the **Node Name** to **ExperienceJEENode**.
  - b. Accept the default value for **Server Name** and **Host name**.
  - c. Click **Next**.

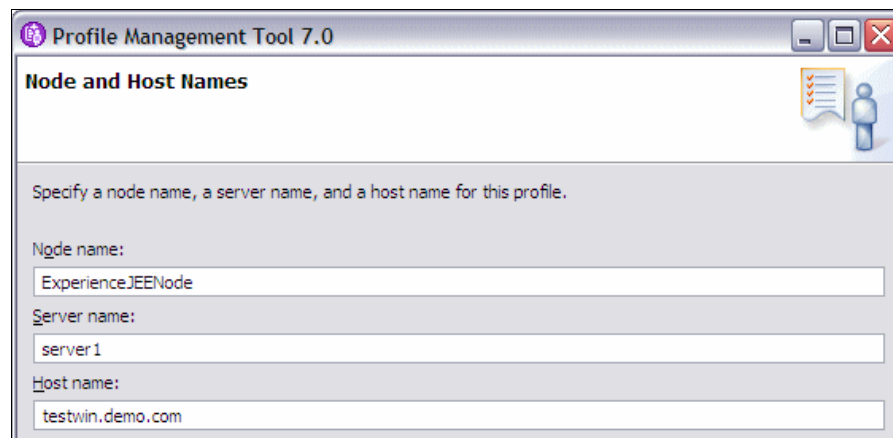


Figure 3-12 Node and Host Names popup

11. At the Administrative Security pop-up:
  - a. Select **Enable administrative security**.
  - b. Set user name to **javaeeadmin**

- c. Set **Password** and **Confirm password** to any password value.
- d. Click **Next**.

#### **Behind the scenes:**

This option configures security for using the WebSphere Application Server administrative interfaces: The Web based WebSphere Administrative Console, the **wsadmin** command line interface, and the Java Management Extensions (JMX) interface. This ensures that only authorized users are able to access the administrative functions.

- ▶ This initial Administrative security setting is implemented using a file based file repository and is separate from any user IDs/passwords in the operating system and/or Lightweight Directory Access Protocol (LDAP) registries.
- ▶ WebSphere Application Server V7.0 allows a progressive level of enabling security:
  - Administrative security only (as configured at this point).
  - Administrative Security and Java EE application security, which are used to control access to Java EE Enterprise Application resources such as Web pages and EJBs.
  - Java 2 security, which protects access to specific system resources and is enabled and disabled independently.

You will implement Java EE application security in Chapter 10, “Implement core security” on page 361.

12. At the Security Certificate (Part I), accept the default settings (create a new default personal certificate and create a new root signing certificate) and click **Next**.

#### **Behind the scenes:**

Users typically configure the profile to use unique certificates to use for the administration and internal usage of the WebSphere Application Server cell, and then post-profile creation import the common certificates that are used in the production environment.

13. At the Security Certificate (Part 2), accept the default settings and click **Next**.
14. At the Port Values assignment pop-up (Figure 3-13 on page 68):

- a. Click **Default Port Values** to change all the values on the right back to the defaults.
- b. Click **Next**.

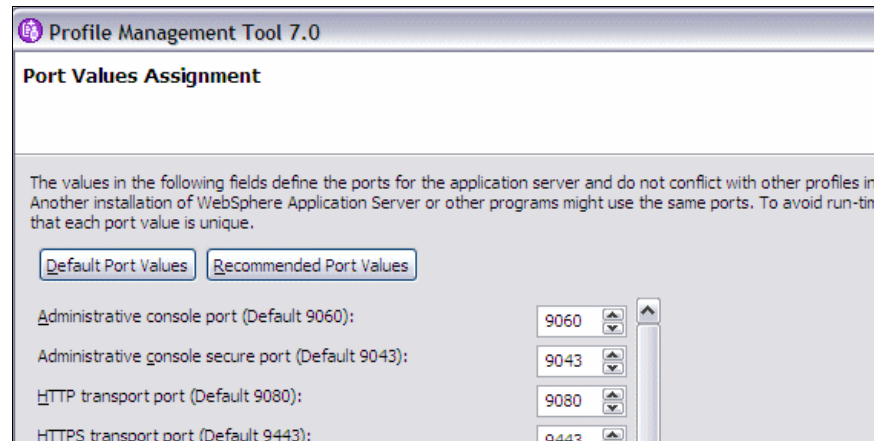


Figure 3-13 Port Values Assignment popup

### Off course?

If you receive the warning message **Activity was defected on these ports: 9060, 9043...** In this case, this indicates that other WebSphere Application Server profiles exist that use these ports.

You would require unique ports if you were planning to run multiple server instances on the same system. Because you will be running a single server, it is simpler to use the defaults.

However, in the context of this book, you will be running with a single active server instance and it should be acceptable to operate with default ports.

15. **Windows** At the Windows Service Definition pop-up, clear **Run the application server process as a Windows service**, and click **Next**.
16. At the Web Server Definition pop-up, leave **Create a Web server definition** DISABLED, and click **Next**.
17. At the Profile Creation Summary pop-up (Figure 3-14 on page 69), click **Create**. This will take a few minutes.



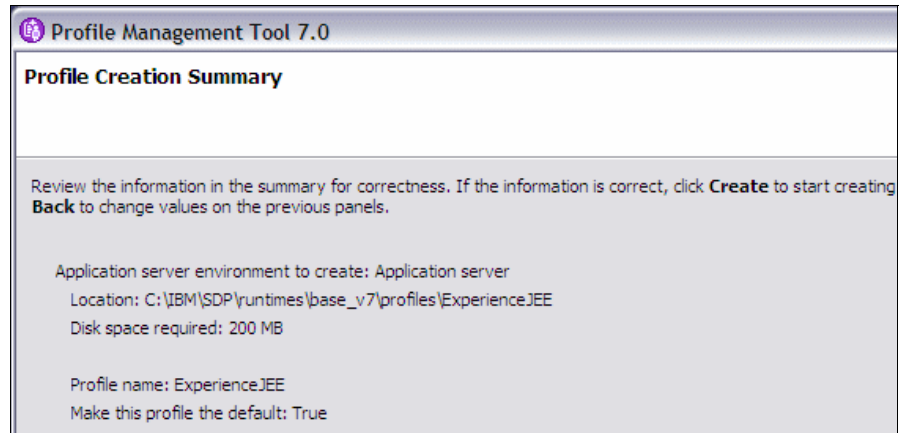


Figure 3-14 Profile Creation Summary popup

18. At the **Profile Creation Complete** pop-up (Figure 3-15), clear **Launch the First steps console**, and click **Finish**.

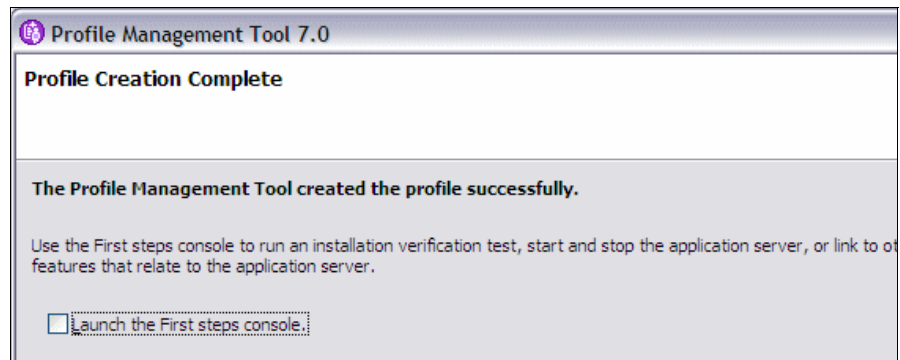


Figure 3-15 Profile Creation Complete popup

### Behind the scenes:

The **First Steps console** provides several tools for WebSphere Application Server 7.0 beginners:

- ▶ **Installation verification**, which executes a test script to verify basic application server functions.
- ▶ **Start the server/Stop the server.**
- ▶ **Administrative console:**  
<http://localhost:9060/ibm/console/>
- ▶ **Profile management tool**, which allows you to create and delete WebSphere Application Server runtime profile definitions.
- ▶ **Information center** available on the Internet:  
<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp>
- ▶ **Migration wizard**, for migrating profiles from earlier versions of WebSphere Application Server.

19. Close the Profile Management Tool pop-up.
20. Back at the **Preferences** pop-up (Figure 3-16 on page 71), note that the new profile is listed. You may need to reselect the WebSphere Application Server installed runtime in order to refresh the profile list..
21. Click **OK** to close the **Preferences** pop-up.

WebSphere Application Server

☐ Allow applications containing errors to be published on a server  
☒ Check for valid profiles and servers during workbench startup

WebSphere Application Server local profile management:

Installed Runtime Name	Installation Directory
WebSphere Application Se...	C:\IBM\SDP\runtimes\base_v7
<div> <div></div> <div></div> </div>	

WebSphere Application Server profiles defined in the runtime selecte:

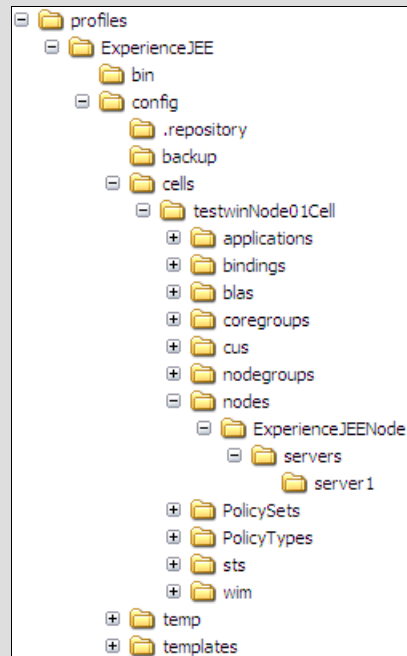
Name	Location
ExperienceJEE	C:\IBM\SDP\runtimes\baserie

Run Profile Management Tool

Delete

Figure 3-16 Preferences popup

### Behind the scenes:



The complete profile definition is created in the file system in the *WAS\_HOME*\profiles directory, as shown in the image to the right.

You are using the profile for a standalone server (in this case called *server1*), but the overall structure is architected to support the complete WebSphere Application Server Network Deployment configuration:

- Node (in this case *ExperienceJEE*Node) is the physical machine that supports zero or more servers.

- Cell (in this case *testwinNode01Cell*) is a logical grouping of nodes that can be centrally administered.

Profiles can be deleted by using the delete button on the workbench WebSphere preferences page or by using the **manageprofiles** (.sh on Linux) command found in the *WAS\_HOME*\bin directory.

```
manageprofiles -delete -profileName ExperienceJEE
```

Note that after running this command you should manually delete the actual profile subdirectory. This is because the command does not completely delete the subdirectory.

## 3.5 Create the ExperienceJEE workspace test server

The workspace can contain the definitions of multiple test servers that identify the runtime environments to which you want to deploy, test, and debug applications.

### Local server versus remote server:

Rational Application Developer allows the definition of local test servers and remote test servers:

- ▶ A local test server runs on the same system as the workbench.
- ▶ A remote test server runs on a different system than the workbench, and its host name is something other than localhost or 127.0.0.1.

From a functional standpoint, a remote test server is almost equivalent to a local test server. The only significant difference is that you cannot start a remote test server from the workbench. Instead, you must start the test server on the remote system, and then you can access it from the workbench.

In this section, we verify that the test server definition was automatically added by the previous section. If it was not, we manually add it here:

1. Switch to the Servers view in the lower-right pane.
2. Right-click the white space, and select **New** → **Server** (Figure 3-17).

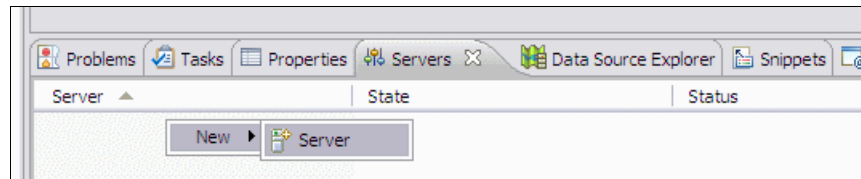


Figure 3-17 Create new server

3. At the New Server/Define a New Server pop-up (Figure 3-18 on page 74):
  - a. Ensure that the server type is set to **IBM** → **WebSphere Application Server v7.0 Server**
  - b. Set Server name to **ExperienceJEE Test Server**.
  - c. Ensure that the Server runtime environment is set to **WebSphere Application Server v7.0**.
  - d. Click **Next**.

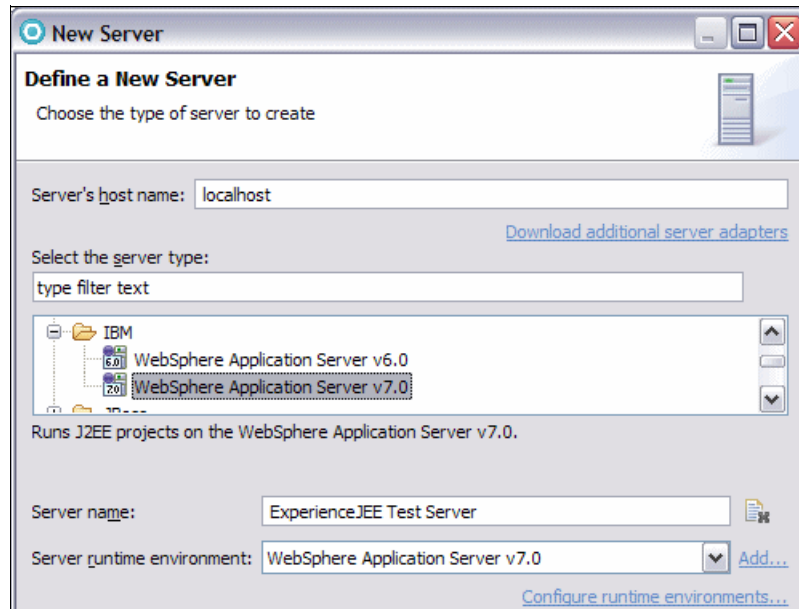


Figure 3-18 New Server/Define a New Server pop-up

4. At the New WebSphere Application Server v7.0 Server pop-up (Figure 3-19 on page 75):
  - a. Ensure that Profile name is set to **ExperienceJEE**.
  - b. Ensure that the **Security is enabled on this server** selection is enabled.
    - Set the username and password to the values you used when creating the server profile in “3.4 Create the ExperienceJEE server profile” on page 62.
  - c. Click **Finish**.

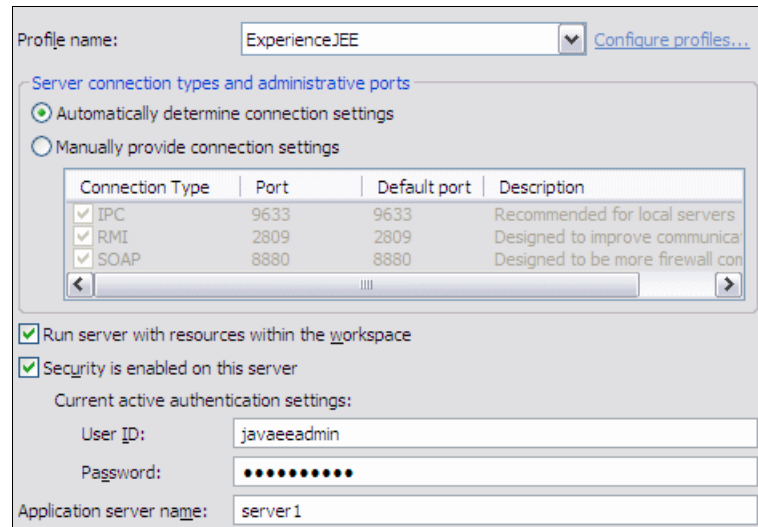


Figure 3-19 New WebSphere Application Server v7.0 Server pop-up

5. The Servers view (Figure 3-20) should now contain the new entry called ExperienceJEE Test Server. Double-click the new server to open in a configuration editor.

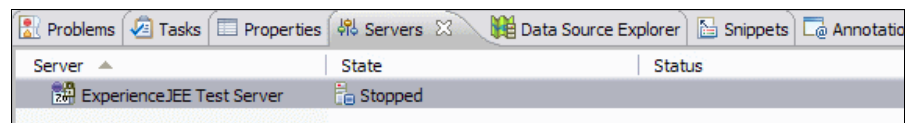


Figure 3-20 Servers view

6. In the resulting ExperienceJEE Test Server editor (Figure 3-21 on page 76):
  - a. In the Publishing section (on the upper right side of the pane), select **Never Publish Automatically**.

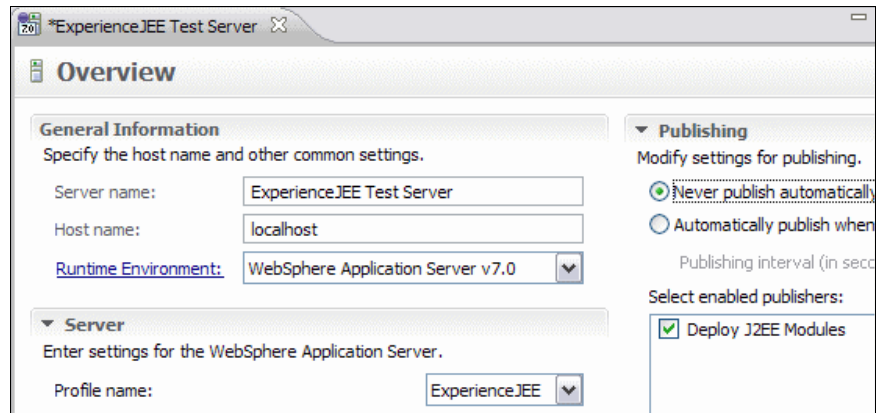



Figure 3-21 ExperienceJEE Test Server editor

- b. Save and close the changes using one of the following methods:
  - i. Click  next to the name, and click **Yes** at the Save Resource pop-up.
  - ii. Ensure that the editor window is active, and select **File** → **Save** (or Ctrl+S) and then **File** → **Close** (or Ctrl+F4).

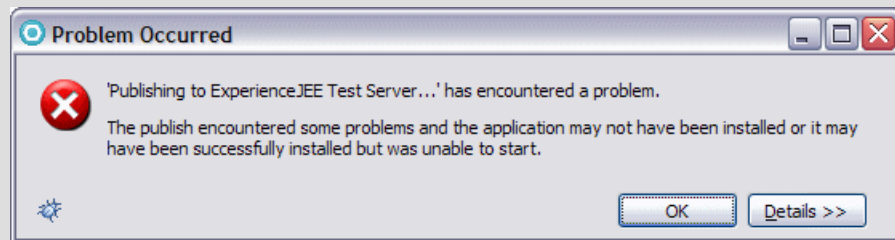


### Behind the scenes:

The server configuration editor contains a combination of configuration options for how the test server is defined to and interacts with the workbench.

### For example, automatic publishing:

- ▶ The default publishing setting is automatic, which instructs the workbench to automatically propagate updates for deployed applications to the test server at the specified interval (the default is 15 seconds). This occurs each time the workbench detects a change to the application. The key benefit of Automatic Publishing is that application server always has the latest version of your application.
- ▶ Automatic publishing does have drawbacks and limitations, such as the performance overhead for large projects, and the situation when your projects contain problems that cause automatic publishing to fail with an error pop-up:



There are at least two different approaches for dealing with these situations:


1. Increase the automatic publishing interval to several minutes, thereby reducing the number of pop-ups.
2. Turn off automatic publishing, which will require you to manually publish the updated application.

Which is the better or best approach? It depends on personal preference. For this book we turn off automatic publishing.

The Servers view shows the current publishing status, and displays **Republish** in the Status field if test server is out of sync with the workspace.

## 3.6 Start the ExperienceJEE Test Server

With the Rational Application Developer workbench and the WebSphere Application Server test environment, most configuration changes are picked up automatically and the test server does not have to be restarted. Therefore, you can start the test server now, and leave it running for most of the subsequent scenarios:

1. In the Servers view (Figure 3-22), select **ExperienceJEE Test Server**. Click the **Start** icon [  ] in the view action bar.

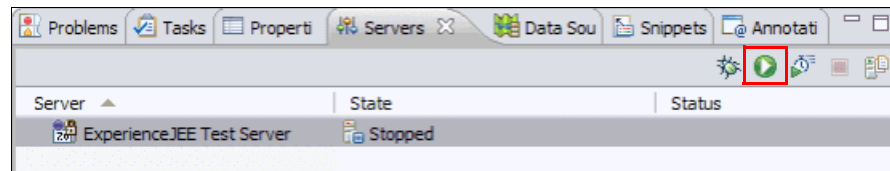



Figure 3-22 Servers view

The server status changes to **Starting....**, and then to **Started** when the startup is complete. Manually switch back to the Servers view to see this final change in status.

If you receive any firewall prompts, allow access by java or javaw.

2. The workbench should automatically switch to the Console view in the lower right (if it does not, click the tab for the Console view), and then you can view the startup messages in the log file (which will show as **ExperienceJEE Test Server (WebSphere Application Server v7.0)**). Verify that startup occurred normally (for example, that there are no red messages), and look for the “Server server1 open for e-business” started message.
3. If you do not see the messages described in step 2, use the Console switch selection [  ] to ensure that the Console view is showing the log for **ExperienceJEE Test Server**.

## 3.7 Import the sample code snippets

The workbench allows you to configure a set of reusable samples or templates called snippets. You can insert these snippets into your artifacts either as a simple cut and paste, or with advanced features such as variable replacement.

In this case, you use snippets as a simple cut and paste mechanism when adding code to various Java classes and Web pages, instead of manually typing in the code or copying from a text file:

1. Switch to the Snippets view (Figure 3-23) in the lower right pane. Select anywhere inside the view, right-click and select **Customize**.

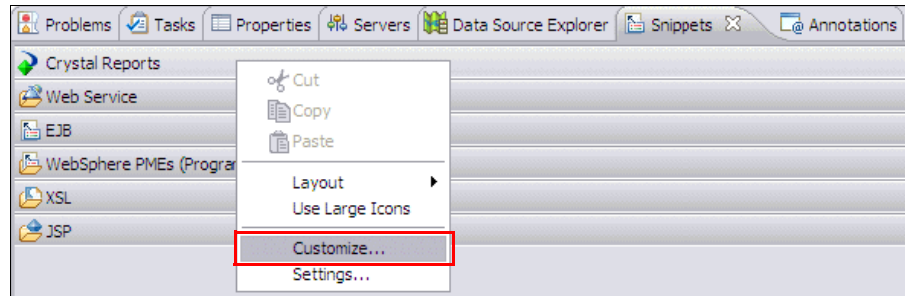


Figure 3-23 Snippets view select Customize

2. At the Customize Palette pop-up (Figure 3-24), click **Import**.

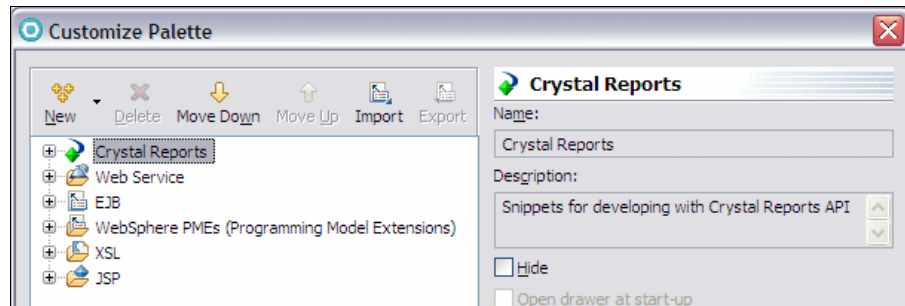






Figure 3-24 Customize Palette pop-up

3. At the directory browser pop-up:
  -  **Windows** Navigate to the directory C:\7827code\snippets, select **experiencejee\_snippets.xml**, and click **Open**.
  -  **Linux** Navigate to the directory \$HOME/7827code/snippets, select **experiencejee\_snippets(.xml)**, and click **OK**.
4. At the Customize Palette pop-up, move the newly created category to the top of the list, by performing the following:
  - a. Scroll to the bottom of the list (if not already there).
  - b. Select the **ExperienceJEE** category.
  - c. Click **Move Up** until the **ExperienceJEE** category is at the top of the list.
5. Expand the **ExperienceJEE** category by clicking the plus expansion symbol located next to it ( on Windows or  on Linux) (Figure 3-25).

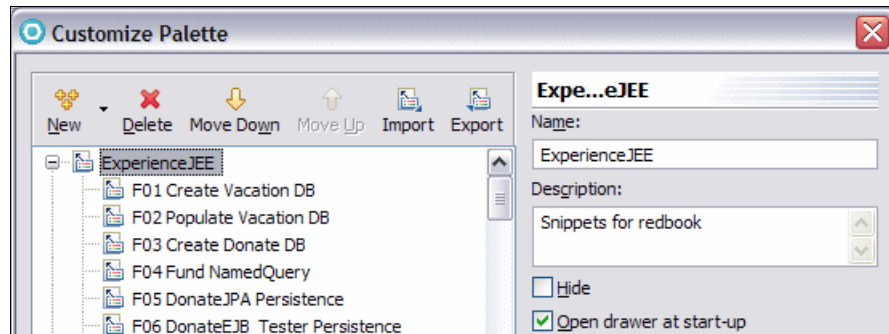


Figure 3-25 Customize Palette - Expand ExperienceJEE

6. Select a snippet and its code is shown on the right side.
7. Click **OK** to close the pop-up.

## 3.8 Workspace preferences

There are a few other preferences that can be set for our advantage.

1. In the workbench action bar, click **Window** → **Preferences**.
2. At the resulting Preferences pop-up (Figure 3-26 on page 80):
  - a. Select **General** → **Editors** → **Text Editors**, and select **Show line numbers**. This setting is practical because error messages sometime refer to line numbers.

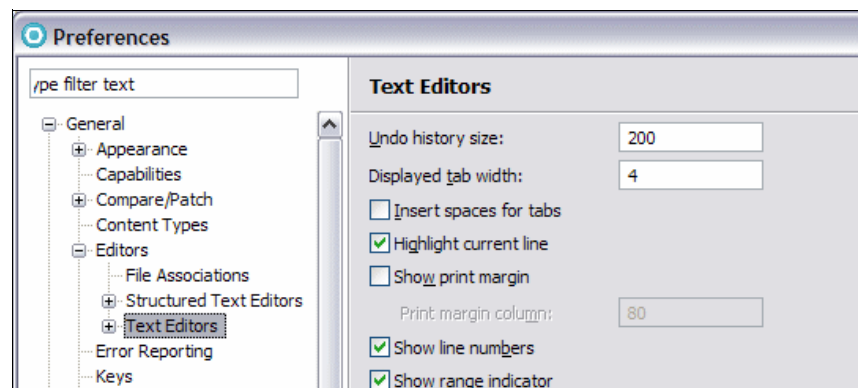


Figure 3-26 Preferences pop-up Text Editor selection

- b. Select **Java** → **Code Style** → **Formatter** to see the preferred formatting style for Java code. Note that you can tailor your own style as well.
- c. Click **OK** when finished to close the Preferences pop-up.

## 3.9 Explore!

Figure 3-27 shows the various integrated development environments that IBM provides for development of J2EE and Java EE applications.

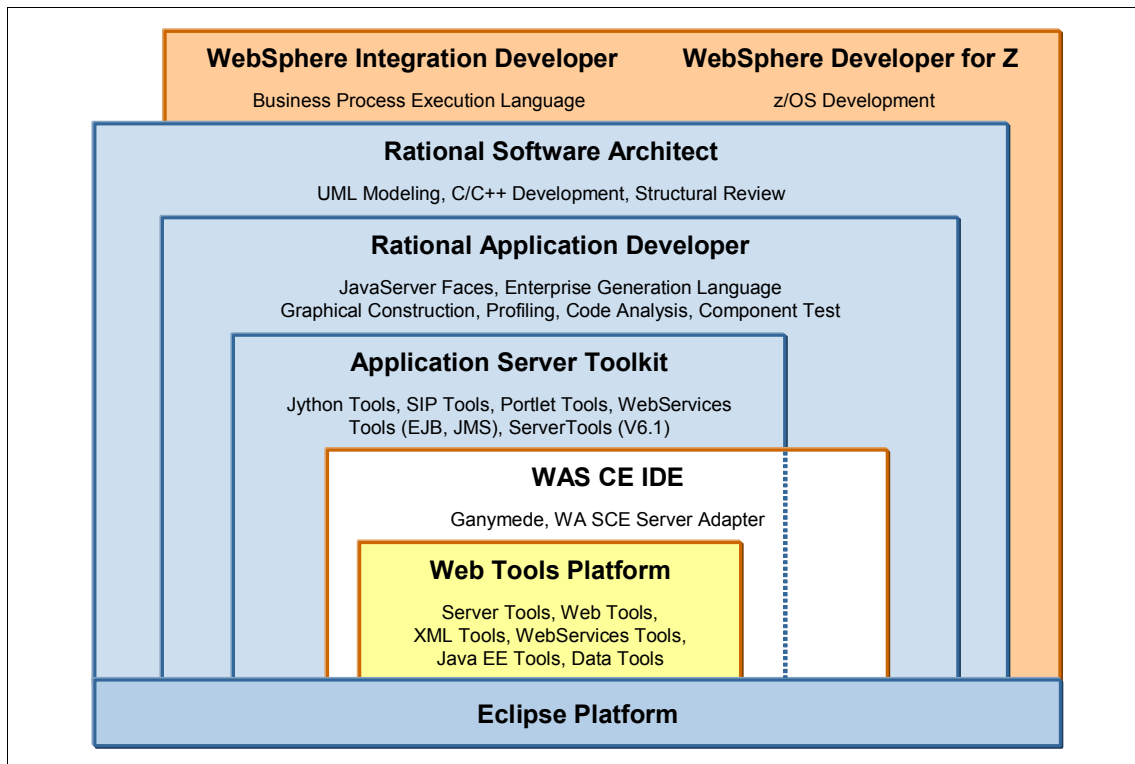


Figure 3-27 IBM integrated development environments

The Rational Application Developer workbench in this chapter provides an Eclipse-based development platform for WebSphere Application Server V7.0. However, just as IBM has other Java runtime environment with WebSphere

Application Server v6.x and WebSphere Application Server Community Edition, IBM also has other Eclipse-based development products to support WebSphere Application Server v6.x and v7.0.

- ▶ **WebSphere Application Server Toolkit v6.1** (AST) provides similar functions as Rational Application Developer, but for the basic WebSphere Application Server v6.x:

- AST is built on top of WTP.
- AST includes Portlet tools and Jython administrative scripting.

**Note:** AST is not available for WebSphere Application Server v7.0, which is shipped with a copy of Rational Application Developer Assembly and Deploy, which provides support for creating, building, testing, and deploying J2EE 1.4 and Java EE 5 applications.

- ▶ **Rational Software Architect v7.0 and v7.5** provide UML based modeling and design tools intended for use by non-programmers such as architects and systems analysts. The models can be used to generate the skeleton Java EE artifacts that then can be completed and extended by Rational Application Developer.
- ▶ **WebSphere Integration Developer v7.0** provides for the development and testing of non-Java EE artifacts such as Enterprise Service Bus (ESB) mediations and WS-BPEL based workflows.

Collectively these products are all based off the same core foundation of Eclipse, the Web Tools Platform (WTP), and targeted server adapters. Thus, the general skills and concepts learned in Rational Application Developer can be leveraged when using these other products.

One key consideration is that these products are based on different versions of Eclipse (v3.2, v3.3, and v3.4) and the WTP (v1.5, v2.0, v3.0). This reflects the reality of the Eclipse environment, where new releases and capabilities are being released each year. 18 month old products (such as the WebSphere Application Server Toolkit v6.1) can easily be two levels behind the latest releases of the underlying open source packages.

The positive side is that these releases represent evolutionary changes and not revolutionary changes. The core functions and capabilities are the same across releases, and are changed to provide improved usability, improved features, and to support the latest technical specification. Someone using Eclipse v3.2 with WTP v1.5 should be able to easily ramp up on using Eclipse v3.4 with WTP v3.0.

Additional Explore resources are available at the following Web sites:

- ▶ WebSphere Application Server V7.0 Announcement letter:

[http://www.ibm.com/common/ssi/rep\\_ca/6/897/ENUS208-266](http://www.ibm.com/common/ssi/rep_ca/6/897/ENUS208-266)

- ▶ IBM developerWorks: Recommended reading list: J2EE and WebSphere Application Server:  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0305\\_issw/recommendedreading.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0305_issw/recommendedreading.html)
- ▶ Rational home page, Rational Application Developer, and Rational Software Architect:  
<http://www.ibm.com/software/rational>  
<http://www.ibm.com/software/awdtools/developer/application/index.html>  
<http://www.ibm.com/software/awdtools/architect/swarchitect/index.html>
- ▶ IBM developerWorks Rational home page:  
<http://www.ibm.com/developerworks/rational>
- ▶ WebSphere Integration Developer home page:  
<http://www.ibm.com/software/integration/wid/>







## Prepare the legacy application

In this chapter, we configure the initial application database that will be used in the Experience Java EE! scenario.

## 4.1 Learn!

Derby is a Java-based “load and go” database that supports standard structured query language (SQL) and JDBC. It provides a full-featured, robust, small-footprint database server that is simple to deploy and that reduces the cost of applications.

Derby does not have to be installed separately because it is included in both the Rational Application Developer workbench and in the embedded WebSphere Application Server test environment.

Derby contains the following two different run-time versions:

- ▶ **Embedded** provides direct access to a database to a single user. This version provides simple *load and go support*: Point the JDBC drivers to the database, and invoke the access. It also has low overhead because it runs in the same JVM as the caller.

The primary disadvantage of the embedded is that you cannot have concurrent access (for example, by both the test server running in one JVM and the workbench running in another JVM). If you want to see if an entity updated a database record, you must first shutdown the test server, before you can view the database with the workbench.

- ▶ **Network Server** provides network access to a database to multiple users for concurrent operation. This version requires that an administrator configure and start the database server. The database client points the JDBC drivers to this network location and database name, and then invokes the access. This version runs in a separate JVM.

For the purpose of this book, Derby allows you to develop and test an application that uses a database without installing a full-function database, such as Oracle Database or DB2.





Derby was seeded with code submission from IBM based on a product feature called Cloudscape.

Additional Learn resources are available at the following Web sites:

- ▶ Apache Derby home page:  
<http://db.apache.org/derby>
- ▶ IBM Cloudscape Information Center (which actually documents the Derby database):  
<http://publib.boulder.ibm.com/infocenter/cscv/v10r1/index.jsp>

## 4.2 Start the Derby Network Server

WebSphere Application Server does not automatically start the Derby Network Server. Therefore, you must manually start the Derby Network Server as described below.

1. From a command prompt, change to the Derby binary directory for the network server:
  -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
  -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`
2. Start the Derby Network Server:
  -  `startNetworkServer.bat`
  -  `startNetworkServer.sh`
3. The Derby Network Server will start in the command window and remain running until terminated or the system is rebooted.

Security manager installed using the Basic server security policy.  
Apache Derby Network Server - 10.3.3.1 - (765035) started and ready to  
accept connections on port 1527 at 2009-12-17 15:14:43.511 GMT

### Behind the scenes:

The startNetworkServer command actually is a shell for the general purpose Derby networkServerControl command which provides an extended set of control, configuration, and status options. Run networkServerControl with a help flag (“?”) to see a full list of options.

The Derby log file (C:\IBM\SDP\runtimes\base\_v7\derby\derby.log in Windows or /opt/IBM/SDP/runtimes/base\_v7/derby/derby.log in Linux) contains additional status information that is not shown in the command window used above. The following log file example was the result of starting the Derby Network Server, connection to the Vacation data source (which you will create in “4.4.1 Create the Vacation Derby database” on page 90), and then disconnecting from the data source:

```
Apache Derby Network Server - 10.3.3.1 - (765035) started and ready to
accept connections on port 1527 at 2009-12-22 15:53:23.914 GMT
-----
2009-12-22 15:53:51.223 GMT:
  Booting Derby version The Apache Software Foundation - Apache Derby -
10.3.3.1 - (765035): instance c013800d-0125-b717-bca2-ffff942fe944
on database directory C:\IBM\SDP\runtimes\base_v7\derby\Vacation

Database Class Loader started - derby.database.classpath=''
2009-12-22 15:53:51.824 GMT Thread[DRDAConnThread_4,5,main] (DATABASE
= Vacation), (DRDAID = {2}), Apache Derby Network Server connected to
database Vacation;create=true

2009-12-22 15:57:39.722 GMT:
Shutting down instance c013800d-0125-b717-bca2-ffff942fe944
-----
```

## 4.3 Configure the Derby JDBC driver

In this section we update the workbench Data Management definition to reference the Derby JDBC drivers contained in the workbench. Note that this could also be configured at step 3 in “4.3 Configure the Derby JDBC driver” on page 88, but we do this first to shorten the overall length of 4.3.

1. In the workbench action bar, select **Window** → **Preferences**.
2. At the Preferences pop-up, on the left scroll down and select **Data Management** → **Connectivity** → **Driver Definitions**.
3. On the right at the resulting Driver Definitions pane:

- Next to Filter, use the pull-down to select **Derby**. Note that you may need to scroll the window to the right to see the pull down tab.
- Under the Name column, select **Derby 10.2 - Derby Client JDBC Driver Default** (Figure 4-1) and click **Edit**.

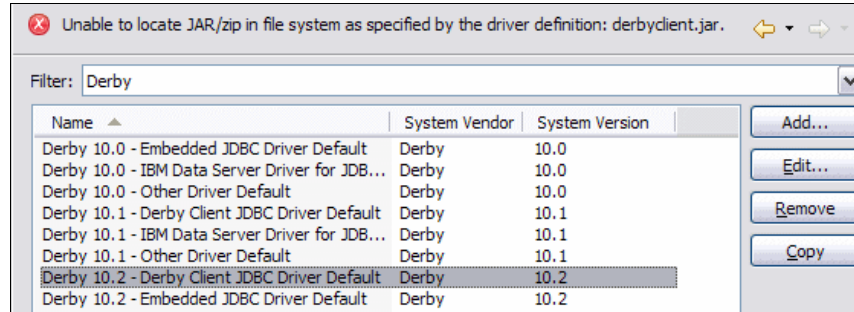


Figure 4-1 Select the Derby Client JDBC Driver

#### Behind the scenes:

The error message at the top of the screen indicates that the database driver definition has not been configured. You will configure this in the following steps, and the error message should be resolved by step 5 on page 90.

- At the resulting Edit Driver Details pop-up:
  - Switch to the Jar List tab.
  - Under Driver files, select **derbyclient.jar** and click **Edit JAR/zip**.
  - At the directory pop-up, select the following file:
    - **Windows**  
C:\IBM\SDP\runtimes\base\_v7\derby\lib\derbyclient.jar
    - **Linux**  
/opt/IBM/SDP/runtimes/base\_v7/derby/lib/derbyclient.jar
  - Verify that the Jar List tab displays the updated Driver file (Figure 4-2 on page 90), and click **OK** to close.

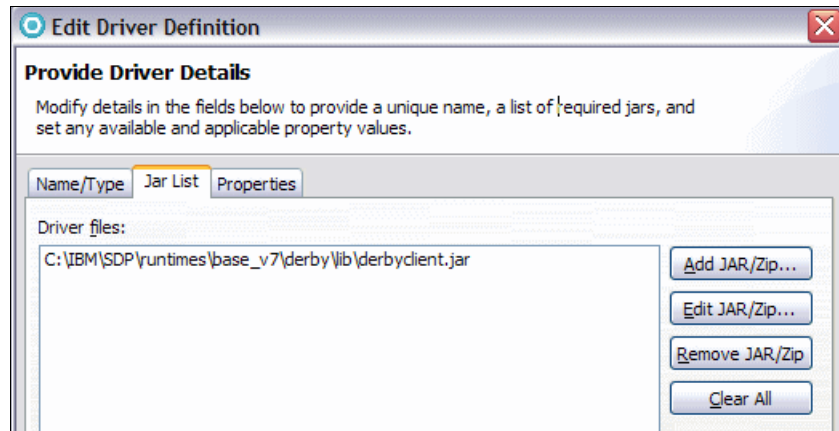


Figure 4-2 Jar list


5. Back at the Preferences pop-up, note that the Unable to locate Jar/Zip... error message has been resolved, and click **OK** to close.

## 4.4 Create and configure the Vacation database

The Vacation database simulates the legacy application, and this section you create the database and configure the Rational Application Developer workbench and the WebSphere Application Server test environment to access the database.

### 4.4.1 Create the Vacation Derby database

Derby does not have a graphical administration tool, so instead we use the data tooling in the workbench to create the database.

1. Switch to the Database Development perspective:
  - Click the Select Perspective icon [  ] located at the upper right, and select **Other** from the pull-down.
  - At the Open Perspective pop-up, select **Database Development** and click **OK**.

### Behind the scenes:

Perspectives are tailored for user roles. Typically a database specialist would work with databases in the Database Development perspective, whereas a Java EE developer would work in the Java EE perspective.

It is possible to add views that are displayed in one perspective to another perspective, where that view is not open by default. Also, users sometimes close views by mistake. Here are instructions on how to deal with views:

- ▶ To add a view to a perspective, in the workbench action bar, select **Window** → **Show View** → **Other**. Navigate to the view, and click **OK**.

For example, if the Data Source Explorer view is not visible, at the Show View pop-up, select **Data** → **Data Source Explorer**, and click **OK**.

- ▶ To reset a perspective to display the default set of views, select **Window** → **Reset Perspective**.
- ▶ You can also save the current combination of panes and views by selecting **Window** → **Save Perspective As**. You can save as a new perspective or change an existing perspective.

2. In the Data Source Explorer view on the left side of the workbench Database Development perspective, select **Database Connections**, right-click and select **New....**

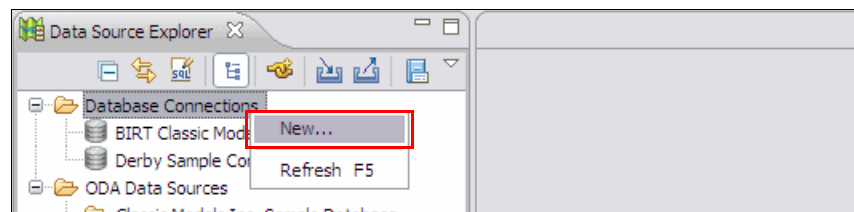


Figure 4-3 Create a new database connection

3. At the New Connection Page pop-up:
  - Uncheck **Use default naming convention**.
  - Set Connection name to **Vacation**.
  - Set Select a database manager to **Derby**
  - Set the JDBC driver to **Derby 10.2 - Derby Client JDBC Driver Default**.
  - Under Properties /General:

- Set Database to **Vacation**.
- Set Host to **localhost**.
- Set Port number to **1527**.
- Set Username to **vacation**.
- Set Password to any value (such as vacation)
- Enable **Create Database (if required)**
- Enable **Save password**.

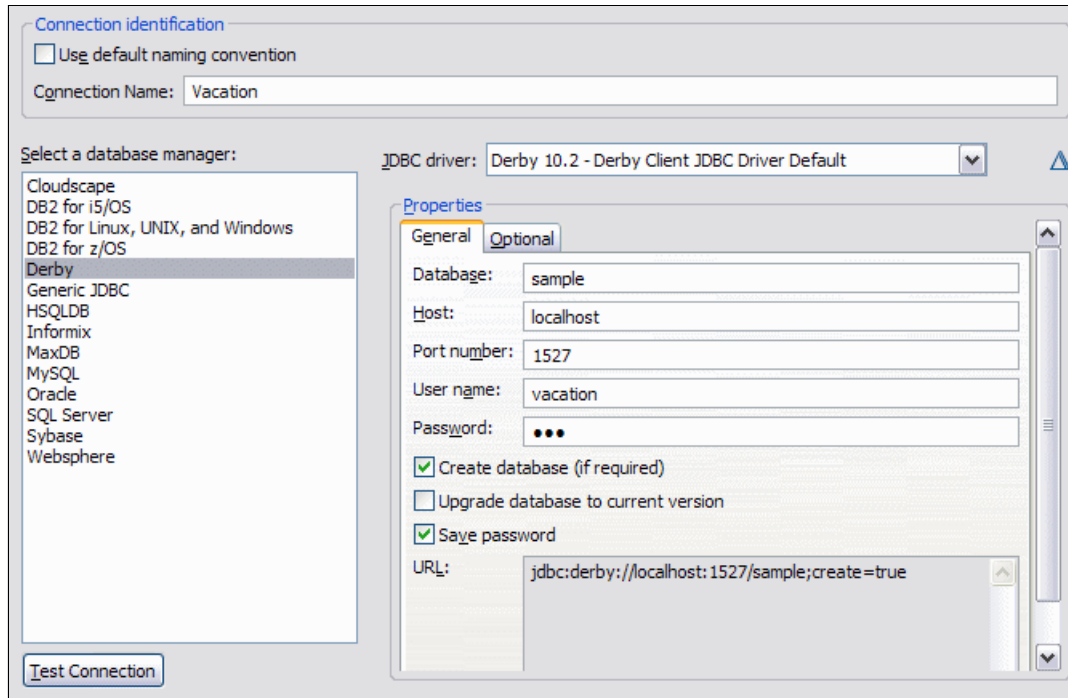


Figure 4-4 Define the new connection

### Behind the scenes:

These fields combine to form the URL string that is used when connecting to the Derby Network running on port 1527:

```
jdbc:derby://localhost:1527/Vacation;create=true:
```

Recall that you started this in “4.2 Start the Derby Network Server” on page 87.



- Click **Test Connection** (bottom right) to verify that the database properties were correctly specified. At the Success pop-up, click **OK**.

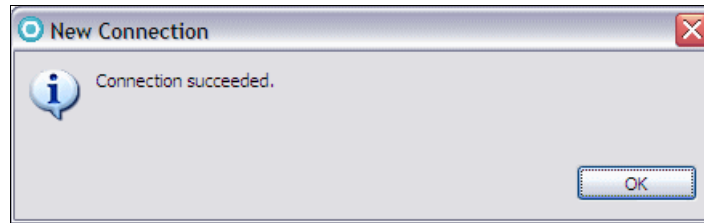
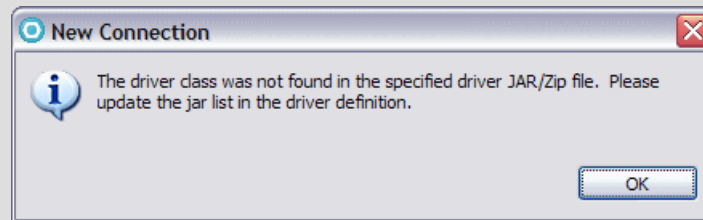


Figure 4-5 Connection succeeded

- Click **Finish**.

#### Off Course?:

If you receive the following pop-up you did not properly configure the driver definition in “4.3 Configure the Derby JDBC driver” on page 88.



**Behind the scenes:** The **create=true** parameter on the URL connection string instructs the Derby Server to automatically create the database. By default the database is created in the subdirectory `C:\IBM\SPD\runtimes\base_v7\derby\Vacation`.

4. The Data Source Explorer view contains the Vacation database connection, initially connected.

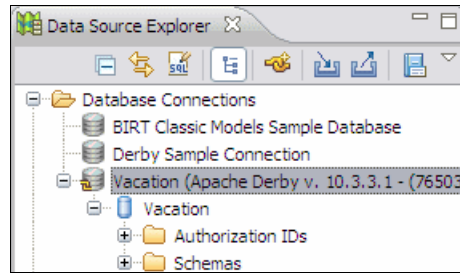


Figure 4-6 Vacation database connection has been established

### Alternatives:

This step uses the **Create the database if required** option in the New Connection wizard to create the database. You can also create the database externally using the Derby command line interface or using the workbench database management function:

- From the Derby **ij** command line  
(C:\IBM\SDP\runtimes\base\_v7\derby\bin\networkServer\ij.bat),  
run the following commands to create the database:  
  

```
connect 'jdbc:derby://localhost:1527/vacation;create=true';
exit;
```

## 4.4.2 Create the employee table in the Vacation database

In this section we use the data tooling in the workbench to create and populate the EMPLOYEE table that contains the application data:

1. In the workbench action bar, select **Window** → **Show View** → **Other**.
  - At the Show View pop-up, select **General** → **Snippets** (or start typing snippet to find the view).
  - Click **OK** and the Snippets view is visible in the lower right pane.

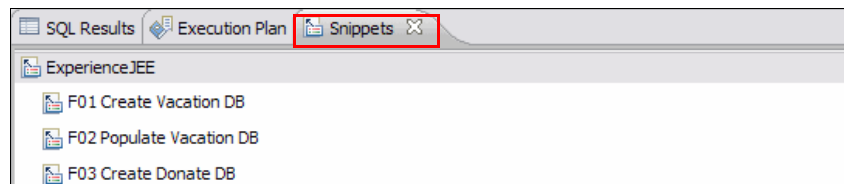


Figure 4-7 Snippets view

2. In the workbench action bar, click the **SQL Scrapbook** icon [  ].

**Alternative:** You can also open a SQL Scrapbook from the Data Source Explorer by selecting the Vacation database connection, right-clicking and selecting **Open SQL Scrapbook**.

3. In the resulting SQL Scrapbook 0 editor, ensure that the following fields are set in the Connection Profile
  - Type: **Derby\_10.x**
  - Name: **Vacation**
  - Database: **Vacation**
4. In the SQL Scrapbook 0 editor:
  - Select the first line of the editor
  - In the Snippets view (lower right), expand the **ExperienceJEE** category, and double-click on **F04.1 Create Vacation DB** to insert the statements into the editor.

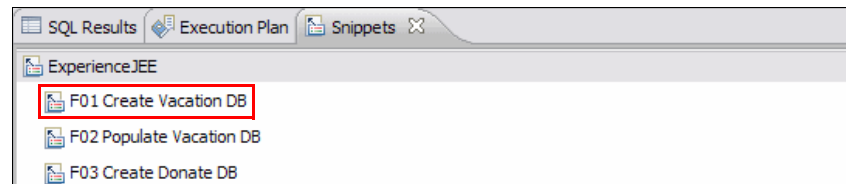


Figure 4-8 Open the snippet

- The SQL scrapbook now contains the SQL statements that create the table and the primary key.
- Select anywhere on the editor, right-click and select **Execute All**.

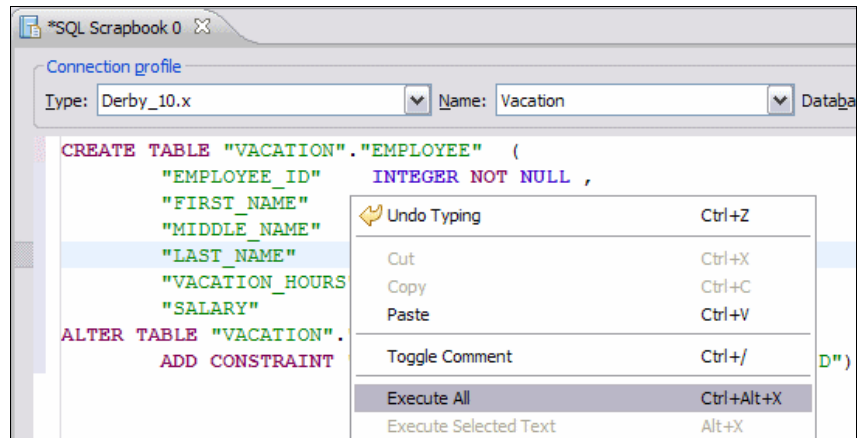


Figure 4-9 Execute the statements

**Off course?** If you do not see the Execute All, option then the Vacation data source is not connected or you did not set the connection name to Vacation in the scrapbook page.

5. The workbench executes the statements and opens an SQL Results view in the lower pane to display the output messages for the commands (create table, alter table, create records).

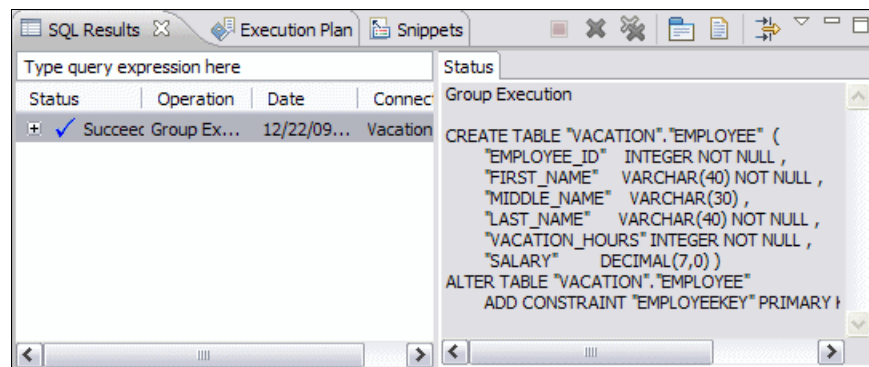


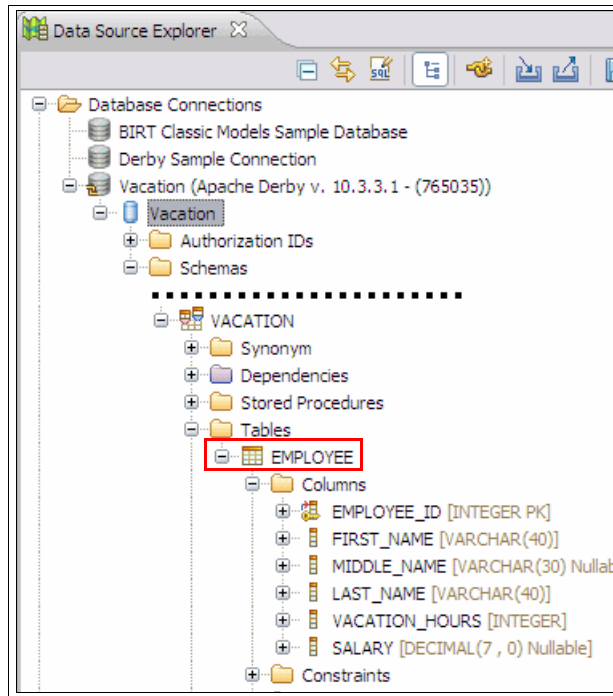
Figure 4-10 SQL results

### Behind the scenes:

The commands are standard DDL (data definition language) statements that create the EMPLOYEE table, and define the primary key.

After executing the DDL statements the EMPLOYEE table is defined in the Vacation database.

6. In the Data Source Explorer view, expand **Database Connections** → **Vacation (Apache Derby v. 10.3.3.1 - (765035))** → **Vacation**, right-click and select **Refresh** to see the new VACATION schema and the EMPLOYEE table under Databases → Vacation (Apache Derby v. 10.3.3.1 - (765035)) → Vacation → Schemas → VACATION → Tables → EMPLOYEE table.



The **Refresh** action reloads the database definition and ensures that the VACATION schema and the EMPLOYEE table appear.

Figure 4-11 Display the Vacation schema and Employee table

### Alternatives:

As before, you can also create the table through the Derby command line interface or the workbench database management function:

- From the Derby **ij** command line, connect, and run the commands:

```
connect 'jdbc:derby://localhost:1527/Vacation';
run 'createEmployee.sql'
exit;
```

## 4.4.3 Populate the Vacation database

In this section we use the data tooling in the workbench to populate the database table with the initial data:

1. In the SQL Scrapbook 0 editor, delete the existing code, and insert the **F04.2 Populate Vacation DB** snippet from the Snippets view:

```
INSERT INTO "VACATION"."EMPLOYEE" ( "EMPLOYEE_ID", "FIRST_NAME",
                                     "MIDDLE_NAME", "LAST_NAME", "VACATION_HOURS")
VALUES (1, 'Charles', 'Patrick', 'Brown', 20);
INSERT INTO "VACATION"."EMPLOYEE" ( "EMPLOYEE_ID", "FIRST_NAME",
                                     "MIDDLE_NAME", "LAST_NAME", "VACATION_HOURS", "SALARY")
VALUES (2, 'Neil', '', 'Armstrong', 50, 100000);
INSERT INTO "VACATION"."EMPLOYEE" ( "EMPLOYEE_ID", "FIRST_NAME",
                                     "MIDDLE_NAME", "LAST_NAME", "VACATION_HOURS", "SALARY")
VALUES (3, 'Rich ', 'ReallyRich', 'Rich', 90, 500000);
```

- Select anywhere on the editor, right-click and select **Execute All**.
2. The workbench executes the statements and displays the results in the SQL Results view.
  3. Close the SQL Scrapbook 0 editor (without saving).
  4. Verify that the data was loaded into the EMPLOYEE table:
    - In the Data Source Explorer view, select **Database Connections** → **Vacation (Apache Derby v. 10.3.3.1 - (765035))** → **Vacation** → **Schemas** → **Vacation** → **Tables** → **Employee**. Right-click and select **Data** → **Sample Contents**.
    - In the resulting SQL Results view in the lower right, switch to the Result1 tab on the right and verify that the three rows were added.

Status	Operation	Date	Connec	EMPLOYEE_ID	FIRST_NAME	MIDDLE_NAME
✓	Succes Group Ex...	12/22/09...	Vacation	1	Charles	Patrick
✓	Succes Group Ex...	12/22/09...	Vacation	2	Neil	
✓	Succes	12/22/09...	Vacation	3	Rich	ReallyRich

Figure 4-12 Three new rows added to the table

### Alternatives:

As in the previous sections, the rows could be added through the Derby `ij` command line or through the workbench database management function.

## 4.5 Create the Donate database

The Donate database represents the new application. You will create the database definitions using the basic steps as you used for the Vacation database above.

However, since this is a database for a new application, there is no existing data to insert into the database. Therefore, you will not create and populate the tables at this time.

1. In the Database Development perspective Data Source Explorer view, select **Database Connections**, right-click and select **New...**
2. At the New Connection Page pop-up:
  - Uncheck **Use default naming convention**.
  - Set Connection name to **Donate**.
  - Set the database manager to **Derby**
  - Set the JDBC driver to **Derby 10.2 - Derby Client JDBC Driver Default**.
  - Under Properties /General:
    - Set Database to **Donate**.
    - Set Host to **localhost**.
    - Set Port number to **1527**.
    - Set Username to **donate**.
    - Set Password to any value.
    - Enable **Create Database (if required)**
    - Enable **Save password**.

- Click **Test Connection** (bottom right) to verify that the database properties were correctly specified. At the Success pop-up, click **OK**.
  - Click **Finish**.
3. The Data Source Explorer view now contains the Donate database connection.

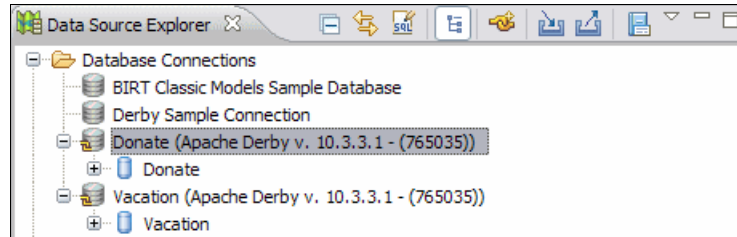


Figure 4-13 The connection to the Donate database has been added

## 4.6 Configure the ExperienceJEE Test Server

The ExperienceJEE Test Server needs to be configured with the proper JDBC provider and JDBC data source to map from the entity EJBs to the Vacation and Donate databases.

### 4.6.1 Start the Administrative Console

The WebSphere Application Server Administrative Console is a browser-based application that allows you to administer the configuration of the individual application server or the entire cell if you are in a cluster.

1. Switch to the Java EE perspective by selecting **Java EE** from the perspective tab in the upper right corner of the workbench.

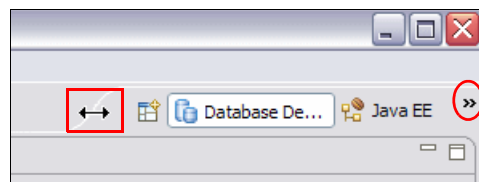




Figure 4-14 Select the Java EE perspective



### Off course:

If you do not see the Java EE perspective in the list there are two possible reasons:

- ▶ The perspective is open but the name is not visible. Two alternatives to address this problem:
  - Drag the left edge of the perspective tab to show more open perspectives. The left edge is shown with the red box above.
  - Double-click on the double-right arrow on the right hand edge of the perspective tab to show a list of additional open perspectives. The double-right arrow is shown with the red circle above.
- ▶ The perspective is not open. These steps will open the perspective:
  - a. Click the Select Perspective icon [  ] located at the upper right, and select **Other** from the pull-down.
  - b. At the Open Perspective pop-up, select **Java EE** and click **OK**.

2. Switch to the Servers view in the lower-right pane, and verify that the ExperienceJEE Test Server is already started—that it has a status of started. If it is not started, select it and click the start icon [  ] in the view action bar to start it.
3. Select the ExperienceJEE Test Server, right-click and select **Administration** → **Run administrative console**.
  - If you receive any browser alerts relating to secure connections or certificates, select the appropriate options to continue.
  - In the Admin Console login screen, set User ID to **javaeeadmin** and Password to the value you set in section 3.4, “Create the ExperienceJEE server profile” on page 62.
  - Click **Log in**.

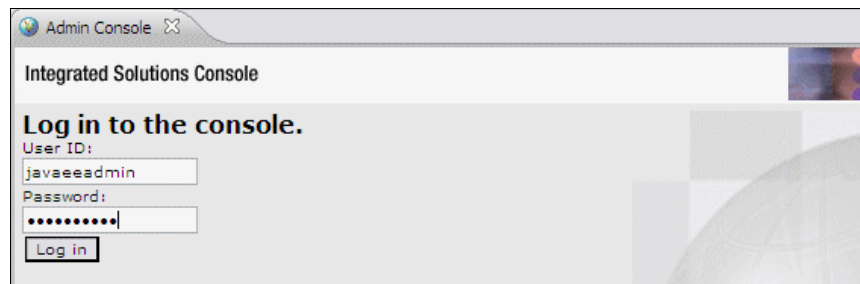


Figure 4-15 Log in to the administrative console

4. The initial page for the **Admin Console** will appear. You may wish to maximize the window by double-clicking the blue **Admin Console** tab.

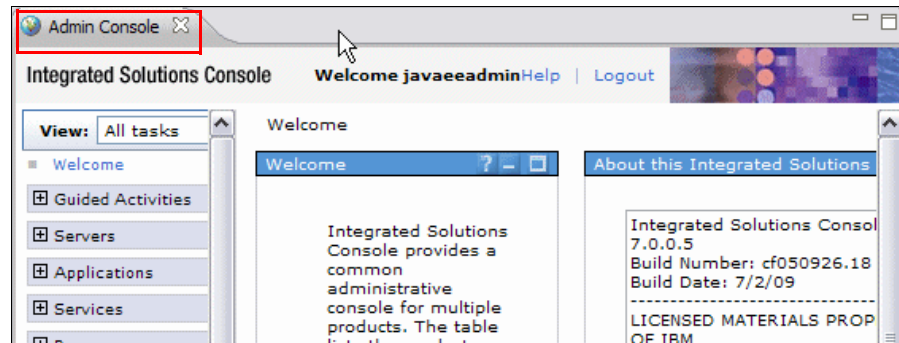


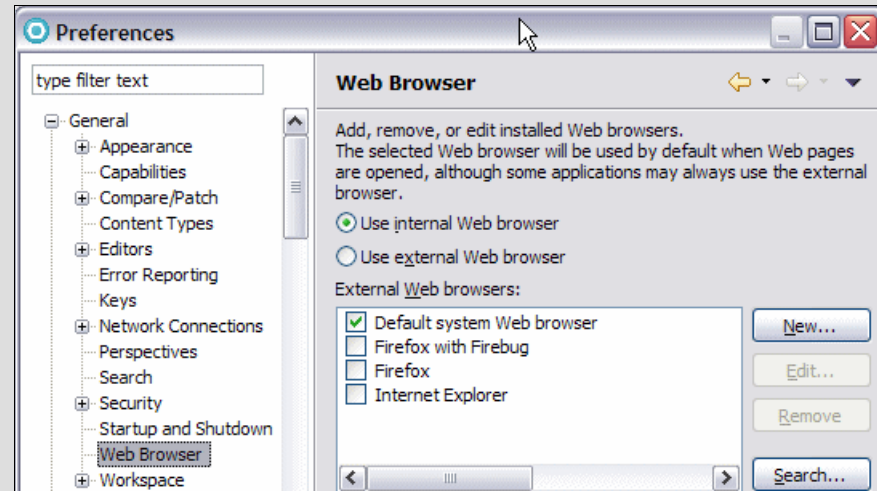
Figure 4-16 Admin console

### Off course?

If the Admin Console appears in an external Web browser, your workspace was not configured to use the internal Web browser as shown in the preceding screen capture.

Rational Application Developer and Eclipse allow you to specify which browser to use. Following are two ways of specifying this:

- ▶ In the workbench action bar, select **Window** → **Web Browser**, and then select the desired value.
- ▶ In the workbench action bar, select **Windows** → **Preferences**. Then, in the resulting **Preferences** pop-up, expand to **General** → **Web Browser**, and select the desired value.



## 4.6.2 Create the JAAS authentication aliases

The Java authentication and authorization service (JAAS) authentication alias provides the runtime user identity that is used when a Java EE artifact, such as an EJB, accesses a Java EE resource such as a JDBC data source.

In the context of JDBC data sources, JPA uses the authentication alias as the default schema for accessing the databases. Therefore, since you have two databases each with a unique schema, you need two authentication aliases.

In this book, two authentication aliases will be used in the JDBC data source definitions which you will define in 4.6.4, “Create the Vacation JDBC data source” on page 112 and 4.6.5, “Create the Donate JDBC data source” on page 118:

- ▶ VacationAlias specifies the **vacation** user will be used when accessing the Vacation database from within the ExperienceJEE Test Server (specifying the vacation user which maps to the vacation schema use for the Employee table you created in 4.4.2, “Create the employee table in the Vacation database” on page 94
- ▶ DonateAlias specifies the **donate** user will be used when accessing the Fund table from within the ExperienceJEE Test Server. This matches the userid (and resulting schema) in the unit test case persistence file which will be used in 5.8.4, “Run the JUnit test” on page 186.

A third authentication alias (ExperienceJEEAlias) specifies the javaeeadmin user that will be used to access messaging resources. You will not utilize this alias until “13.4 Configure the test server for base JMS resources” on page 484 but you configure at this point simply to consolidate the authentication alias definitions at one location in this book.

## Alternatives:

This book uses the JAAS Authentication Alias to manipulate the schema associated with a JPA entity, but there are other approaches:

- Specify the schema within the class:

```
...
@Entity
@Table(name="EMPLOYEE", schema = "VACATION")
public class Employee implements Serializable {
...

```

- Specify the connection user (and as a result the schema) within the persistence.xml file:

```
<persistence-unit name="DonateJPAEmployee">
  <jta-data-source>jdbc/Vacation</jta-data-source>
  <class>vacation.entities.Employee</class>
  <properties>
    <property name="openjpa.ConnectionUserName" value="VACATION" />
  </properties>
</persistence-unit>
```

Which approach is preferred -- class file, persistence.xml file, or JDBC data source configuration? The answer is debatable, and your own preference will depend on whom you believe is the most critical role in ensuring the proper configuration: the java developer (class file), the Java EE packager (persistence.xml file) or the Java EE administrator (JDBC data source configuration)? Probably the most accurate observation is all these roles must be aware of the schema and take appropriate steps to ensure that the proper schema is accessed.

If you do not specify an authentication alias, the EJB will access the datasource as the Primary administrative user name, in this case javaeeadmin (which you defined in 3.4, "Create the ExperienceJEE server profile" on page 62). The result is the access to Employee or Fund will fail because the EJB will look for the table in the javaeeadmin schema instead of the appropriate schema (vacation or donate):

```
Schema 'JAVAEEADMIN' does not exist {SELECT t0.FIRST_NAME, t0.LAST_NAME,
t0.MIDDLE_NAME, t0.salary, t0.VACATION_HOURS FROM Employee t0 WHERE
t0.EMPLOYEE_ID = ?} [code=-1, state=42Y07]
```



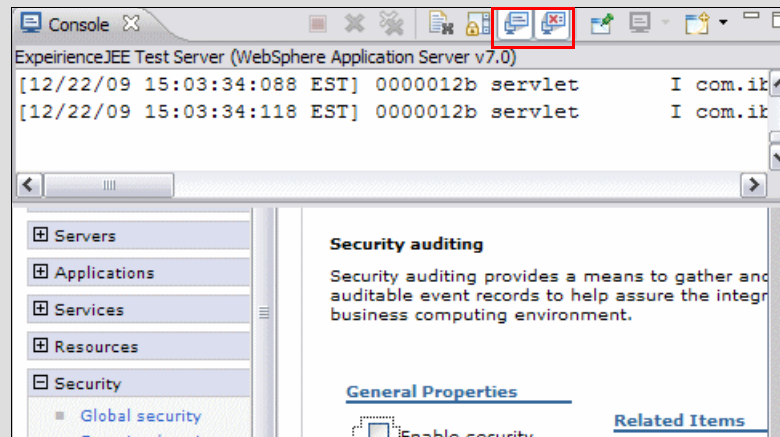
1. In the Admin Console, on the left select/expand **Security** (to change the  to a ) and then select **Global Security** Note that future instructions will describe this shorten this type of description to select **Security** → **Global Security**.



Figure 4-17 Select Global security in the console

## Off Course?

If you had maximized the Admin Console, the Console View by default will automatically pop-up across the top of the Admin Console window to display new messages that are being logged by the ExperienceJEE Test Server.



This pop-up will automatically disappear with your next keystroke, but will continue to re-appear whenever the Console View is updated while you are in full screen mode.

If you find this annoying, you can turn off the pop-up behavior by deselecting the **Show Console When Standard Out [Error] Changes** icons (marked by the red box in the preceding image).

2. On the right, in the Global Security panel, under the **Authentication** section select **Java Authentication and Authorization Service** → **J2C authentication data**.

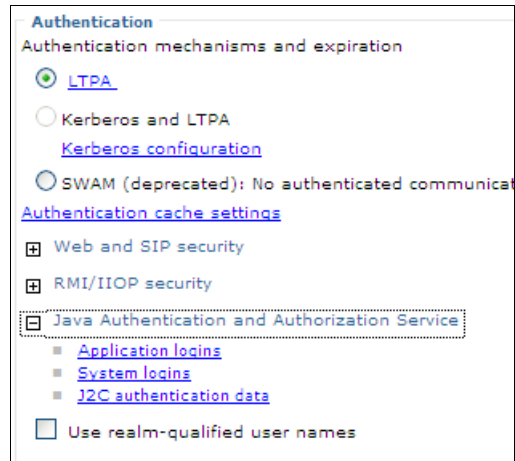


Figure 4-18 Select the J2C authentication data option

3. Create the Vacation authentication alias (for use with the Vacation.Employee table in the Vacation database):
  - On the right, in the Global Security → J2C authentication data panel, click **New**.

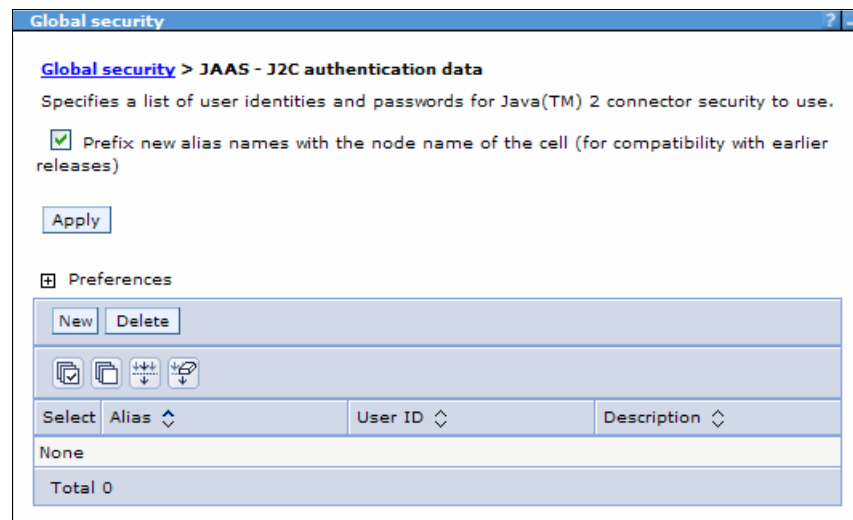


Figure 4-19 Create a new J2C authentication data entry

- On the right, in the Global Security → J2C authentication data → New panel:

- Alias: **VacationAlias**
- User ID: **vacation**
- Password: any value
- Click **Apply**.

The screenshot shows a 'General Properties' dialog box with the following fields:


- \* Alias:** VacationAlias
- \* User ID:** vacation
- \* Password:** \*\*\*\*\*
- Description:** (empty field)

Figure 4-20 J2C authentication data

4. Create the Donate authentication alias (for use with the Donate.Fund table in the Donate database):
  - On the right, in the Global Security → J2C authentication data panel, click **New**.
  - On the right, in the Global Security → J2C authentication data → New panel:
    - Alias: **DonateAlias**
    - User ID: **donate**
    - Password: any value
    - Click **Apply**.
5. Create the ExperienceJEE authentication alias (for use with the messaging resources):
  - On the right, in the Global Security → J2C authentication data panel, click **New**.
  - On the right, in the Global Security → J2C authentication data → New panel:
    - Alias: **ExperienceJEEAlias**
    - User ID: **javaeeadmin**
    - Password: the password defined in “3.4 Create the ExperienceJEE server profile” on page 62
    - Click **Apply**.



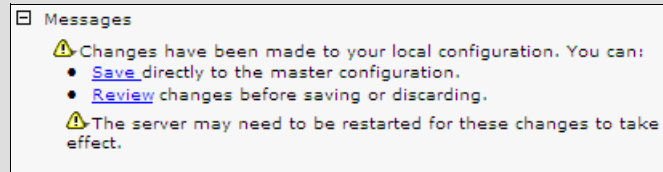
6. Three authentication aliases should now be displayed in the Global Security → J2C authentication panel.



Select	Alias	User ID	Description
<input type="checkbox"/>	<a href="#">ExperienceJEENode/DonateAlias</a>	donate	
<input type="checkbox"/>	<a href="#">ExperienceJEENode/ExperienceJEEAlias</a>	javaeeadmin	
<input type="checkbox"/>	<a href="#">ExperienceJEENode/VacationAlias</a>	vacation	

Figure 4-21 Authentication aliases

**Behind the scenes:** Note the message at the top of the console indicating that the local configuration was changed but that the changes were not saved. Ignore this for now because you save all your changes in a subsequent section.



### 4.6.3 Create the Derby XA JDBC provider

The default runtime environment comes with a non-XA Derby JDBC provider, but you need an XA provider because in subsequent chapters you will create a session EJB that concurrently accesses two separate databases (and this requires XA support).

1. In the Admin Console, on the left select **Resources** → **JDBC** → **JDBC Providers**.

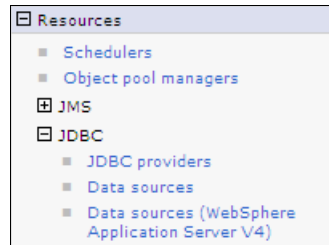


Figure 4-22 List the JDBC providers

2. On the right, in the JDBC providers panel, under Scope use the pull-down to change the scope from **All scopes** to **Node=ExperienceJEENode, Server=server1**.

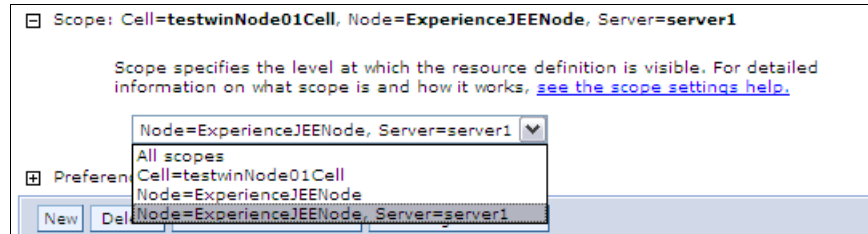


Figure 4-23 Change the scope

### Behind the scenes:

WebSphere Application Server with the Deployment Manager component supports the administration of a group of systems, each running one or more instances.

The file-based configuration (that is, the profile you created in section 3.4, “Create the ExperienceJEE server profile” on page 62) is therefore architected to support such a hierarchy, even when running a single instance:

- ▶ Cell is a grouping of systems (Nodes). The default value is based on the TCP/IP host name for your system and the number of preceding profiles that existed before creating this profile. For example, if the host name is testwin and this is the second profile, the cell name is testwinNode02Cell.
- ▶ Node is an individual system. The default value is based off the host name and the number of other nodes in this cell. Thus, if the host name was testwin, the default value for the first node in the cell would be testwinNode01. You overrode that value when creating the profile and instead used ExperienceJEENode.
- ▶ Server is a unique instance of an application server definition. The default value for the first server on a node (and the value used in the ExperienceJEE profile) is server1.

The configuration for any individual server instance is based on the composite definition (cell, node, server) with the lower definitions taking precedence—meaning the settings for a server overrides the setting for a node.

3. Click **New** to launch the JDBC provider wizard.
4. In the Step 1: Create new JDBC Provider panel:
  - **Database type:** **Derby**
  - **Provider type:** **Derby Network Server Using Derby Client**
  - **Implementation type:** **XA data source**
  - Accept the defaults for **Name** and **Description**.
  - Click **Next**.

→ Step 1: Create new JDBC provider

Step 2: Enter database class path information

Step 3: Summary

Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope

cells:testwinNode01Cell:nodes:ExperienceJEENode:servers:server1

\* Database type

Derby

\* Provider type

Derby Network Server Using Derby Client

\* Implementation type

XA data source

\* Name

Derby Network Server Using Derby Client (XA)

Figure 4-24 Create a new JDBC provider - step 1

- In the Step 3: Summary panel, click **Finish**. Note that we skipped Step 2 because the Derby classpath is already configured.
- Back in the JDBC providers panel, and note the new entry.

New Delete			
<div> <div> <div></div> <div></div> <div></div> <div></div> </div> </div>			
Select	Name	Scope	Description
You can administer the following resources:			
<input type="checkbox"/>	<a href="#">Derby JDBC Provider</a>	Node=ExperienceJEENode,Server=server1	Derby embedded non-XA JDBC Provider
<input type="checkbox"/>	<a href="#">Derby Network Server Using Derby Client (XA)</a>	Node=ExperienceJEENode,Server=server1	Derby Network Server (XA) Provider that uses the Derby Client. This provider is only configurable in version 6.1 and later nodes

Figure 4-25 New JDBC provider

#### 4.6.4 Create the Vacation JDBC data source

The JDBC data source defines the connection to the specific database (in this case the Vacation database created in section 4.4.1, “Create the Vacation Derby database” on page 90).

- In the Admin Console, on the left select **Resources** → **JDBC** → **Data sources**.

2. On the right in the Data sources panel, under Scope use the pull-down to change the scope from **All scopes** to **Node=ExperienceJEENode, Server=server1**.
3. Click **New** to launch the Data sources wizard.
4. In the Step 1: Enter basic data source information panel:
  - Data source name: **Vacation Data source**
  - JNDI name: **jdbc/Vacation**
  - Click **Next**.

<p>→ <b>Step 1: Enter basic data source information</b></p> <p>Step 2: Select JDBC provider</p> <p>Step 3: Enter database specific properties for the data source</p> <p>Step 4: Setup security aliases</p> <p>Step 5: Summary</p>	<p><b>Enter basic data source information</b></p> <p>Set the basic configuration values of a datasource for association with your JDBC provider. A datasource supplies the physical connections between the application server and the database.</p> <p>Requirement: Use the Datasources (WebSphere(R) Application Server V4) console pages if your applications are based on the Enterprise JavaBeans(TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.</p> <p>Scope  <input type="text" value="cells:testwinNode01Cell:nodes:ExperienceJEENode:servers:server1"/></p> <p>* Data source name  <input type="text" value="Vacation Data Source"/></p> <p>* JNDI name  <input type="text" value="jdbc/Vacation"/></p>
--	--

Figure 4-26 Basic data source information

5. In the Step 2: Select JDBC provider panel:
  - Select **Select an existing JDBC provider**.
  - Select **Derby Network Server using Derby Client (XA)** from the pull-down list.
  - Click **Next**.

<p>Step 1: Enter basic data source information</p> <p>→ <b>Step 2: Select JDBC provider</b></p> <p>Step 3: Enter database specific properties for the data source</p> <p>Step 4: Setup security aliases</p> <p>Step 5: Summary</p>	<p><b>Select JDBC provider</b></p> <p>Specify a JDBC provider to support the datasource. If you choose to create a new JDBC provider, it will be created at the same scope as the datasource. If you are selecting an existing JDBC provider, only those providers at the current scope are available from the list.</p> <p><input type="radio"/> Create new JDBC provider</p> <p><input checked="" type="radio"/> Select an existing JDBC provider</p> <p><input type="text" value="Derby Network Server Using Derby Client (XA)"/></p>
--	--

Figure 4-27 Select the JDBC provider

6. In the Step 3: Enter database specific properties for the data source panel:
  - Database Name: **Vacation**
  - Select **Use this data source in container managed persistence (CMP)**.
  - Click **Next**.

Name	Value
* Database name	Vacation

☒ Use this data source in container managed persistence (CMP)

Figure 4-28 Database name

7. In the Step 4: Setup security alias panel,:
  - Under Component-managed authentication alias, select **ExperienceJEENode/VacationAlias** from the pull-down
  - Under XA recovery authentication alias, select **ExperienceJEENode/VacationAlias** from the pull-down.
  - Click **Next**.

Authentication alias for XA recovery  
ExperienceJEENode/VacationAlias

Component-managed authentication alias  
ExperienceJEENode/VacationAlias

Mapping-configuration alias  
(none)

Figure 4-29 Select the security aliases

8. In the Step 5: Summary panel, click **Finish**.
9. Return to the Data sources panel, and note the new entry.

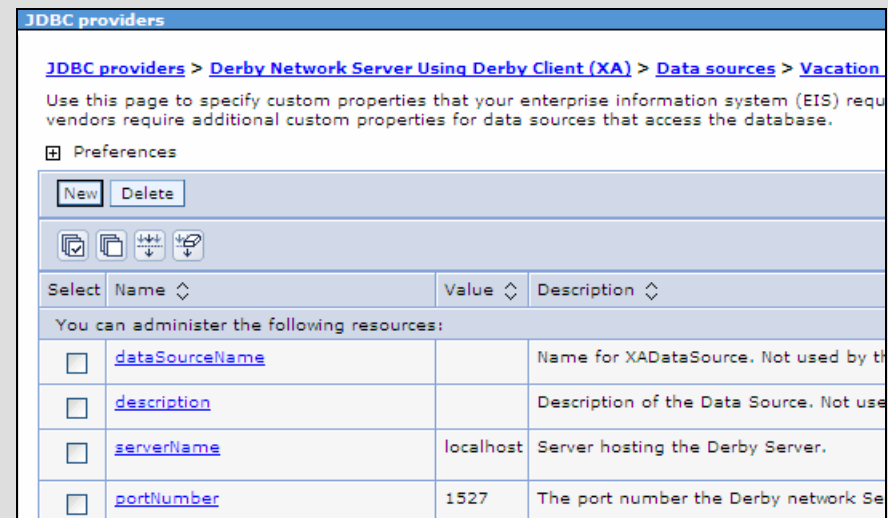
<div> <div>New</div> <div>Delete</div> <div>Test connection</div> <div>Manage state...</div> </div>				
<div> <div> <div>✓</div> <div>📄</div> <div>⬆️⬆️⬆️</div> <div>⬆️</div> </div> </div>				
Select	Name ↕	JNDI name ↕	Scope ↕	Provider
You can administer the following resources:				
<input type="checkbox"/>	<a href="#">Default Datasource</a>	DefaultDatasource	Node=ExperienceJEENode,Server=server1	Derby JD Provider
<input type="checkbox"/>	<a href="#">Vacation Data Source</a>	jndi/Vacation	Node=ExperienceJEENode,Server=server1	Derby Network Server Using Derby Client (X)

Figure 4-30 New data source

## Behind the scenes:

The Derby Network Server data source by default accesses a Derby network server at localhost:1527, so no additional configuration changes are needed. However, if needed, you could view or change properties from the Custom properties page:

- ▶ On the right in the Data sources panel, click **Vacation Data Source**.
- ▶ On the right in the resulting Configuration panel, on the right hand side click **Additional Properties** → **Custom properties**.
- ▶ On the right in the Custom properties panel, note the various default settings, including serverName (localhost) and portNumber (1527).
- ▶ You can change a property by clicking on the name:
  - In the resulting panel, enter the changes and click Apply or OK to save the change.



10. At the top of the panel, in the Messages box that states “Changes have been made to your local configuration....”, click **Save**.

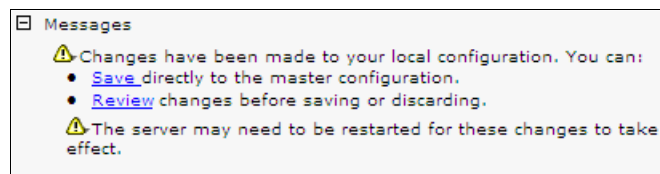


Figure 4-31 Save the changes



### Behind the scenes:

Changes made through the Admin Console are not active until you save the changes and they are written out to the ExperienceJEE profile on disk.

Users have different philosophies on how often to save changes. Some prefer to change frequently (to ensure that updates are not lost in case of failure), while others prefer to wait until all necessary changes are made, thus ensuring that a 'valid' configuration is available.

In this case, you have made all the necessary changes, and in fact you must save before testing the data source that you just created.

#### 11. Test the JDBC configuration:

- Return to the Data sources panel.
- Select the check box next to **Vacation Data Source**.
- Click **Test connection**. The test connection should show the following message:

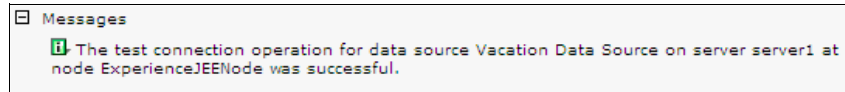


Figure 4-32

#### 12. Leave the Admin Console open for use in the next section.

### Behind the scenes

One advantage of using the Derby Network Server instead of the embedded Derby is that multiple JVMs can concurrently access the database. With the embedded Derby, only a single JVM can access the database.

In this book, you are accessing the databases from two JVMs: the Rational Application Developer workbench and the WebSphere Application Server test environment.

If you had used embedded Derby, and if the workbench database connection was still active, this test connection would have failed with a message similar to the following:

```
The test connection operation failed for data source Vacation
Data source on server server1 at node ExperienceJEE with the
following exception: java.sql.SQLException: Failed to start
database 'C:\ExperienceJEE\Vacation', see the next exception
for details.DSRA0010E: SQL State = XJ040, Error Code = 40,000.
View JVM logs for further details.
```

## 4.6.5 Create the Donate JDBC data source

The JDBC data source defines the connection to the specific database (in this case the Donate database created in 4.5, “Create the Donate database” on page 99).

1. In the Admin Console, on the left select **Resources** → **JDBC** → **Data sources**.
2. On the right in the Data sources panel, under Scope use the pull-down to change the scope from **All scopes** to **Node=ExperienceJEENode, Server=server1**.
3. Click **New** to launch the Data sources wizard.
4. In the Step 1: Enter basic data source information panel:
  - Data source name: **Donate Data source**
  - JNDI name: **jdbc/Donate**
  - Click **Next**.
5. In the Step 2: Select JDBC provider panel:
  - Select **Select an existing JDBC provider**.

- Select **Derby Network Server using Derby Client (XA)** from the pull-down list.
  - Click **Next**.
6. In the Step 3: Enter database specific properties for the data source panel:
- Database Name: **Donate**
  - Select **Use this data source in container managed persistence (CMP)**.
  - Click **Next**.
7. In the Step 4: Setup security alias panel,:
- Under Component-managed authentication alias, select **ExperienceJEENode/DonateAlias** from the pull-down
  - Under XA recovery authentication alias, select **ExperienceJEENode/DonateAlias** from the pull-down.
  - Click **Next**.
8. In the Step 5: Summary panel, click **Finish**.
9. Return to the Data sources panel, and note the new entry.
10. At the top of the panel, in the Messages box that states “Changes have been made to your local configuration...”, click **Save**.
11. Test the JDBC configuration:
- Return to the **Data sources** panel.
  - Select the check box next to **Donate Data Source**.
  - Click **Test connection**. The message box should indicate a successful connection.
12. Click **Logout** located at the top of the Admin Console.
- After the log out completes, close the Admin Console.

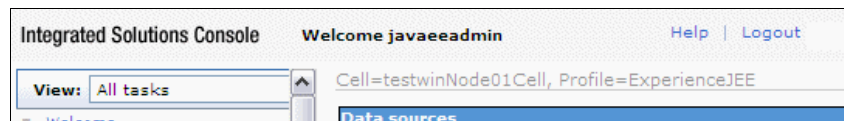


Figure 4-33 Log out

## 4.7 Explore!

Additional Explore resources are available at the following Web site:

- Derby administration guide (containing information about the Network Server):

<http://db.apache.org/derby/docs/dev/adminguide>



## Part 2

# Core Java EE application

In Part 2, we describe how to implement a traditional Java EE application, starting with a single back-end relational database called *Vacat ion*, which we created in Chapter 4, “Prepare the legacy application” on page 85, and ending with a multi-tier application that implements the core Java EE elements of persistent entities, session EJBs, and JSPs, supporting both Web browsers and a thin Java client.





## Create the JPA entities

In this chapter, we create two JPA entities that coordinate and mediate access with two Derby databases:

- ▶ The first entity, `Employee`, is created in a *bottom-up scenario* from the existing `EMPLOYEE` table in the `Vacation` database.
- ▶ The second entity, `Fund`, is created in a *top-down scenario* and a matching `FUND` table in the `Donate` database is created for it.

## 5.1 Learn!

Figure 5-1 shows the WebSphere Application Server CE location in the Presentation and Business Logic layer.

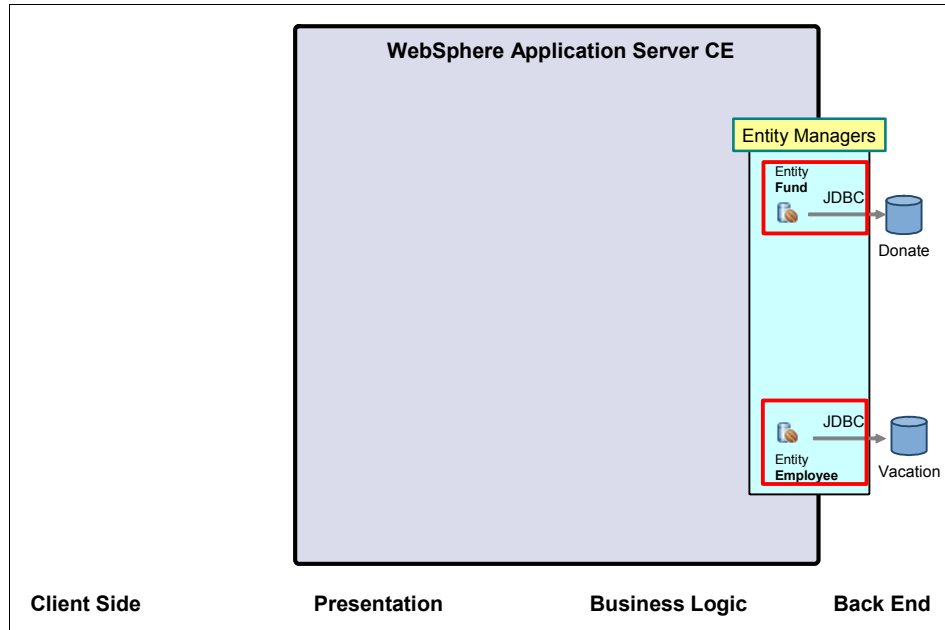


Figure 5-1 Donate application: JPA entities and database mapping

The Learn section of this chapter is an extract from the IBM Redbooks publication, *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611, Chapter 2, Introduction to JPA.

The persistence layer in a typical Java EE application that interacts with a relational database has been implemented over the last years in several ways:

- ▶ EJB 2.x entity beans
- ▶ Data access object (DAO) pattern
- ▶ Data mapper frameworks (such as iBatis)
- ▶ Object-relational mapping (ORM) frameworks both commercial (such as Oracle Toplink) or from the open source world (such as Hibernate)



Data mapper and ORM frameworks gained a great approval among developers communities, because they give a concrete answer to the demand of simplification of the design of the persistence layer, and let developers concentrate on the business related aspects of their solutions.

However, even if these frameworks overcome the limitations of EJB 2.x CMP entity beans, one of the common concerns related to the adoption of such frameworks was that they were not standardized.

One of the main innovative concepts introduced by EJB 3.0 is the provisioning of a single persistence standard for the Java platform that can be used in both the Java EE and Java SE environments, and that can be used to build the persistence layer of a Java EE application. Furthermore, it defines a pluggable service interface model, so that you can plug in different provider implementations, without significant changes to your application.

The Java Persistence API provides an object relational mapping facility to Java developers for managing relational data in Java applications. Java persistence consists of three areas:

- ▶ Java Persistence API
- ▶ Object-relational mapping metadata
- ▶ Query language

### 5.1.1 Entities

In the EJB 3.0 specification, entity beans have been substituted by the concept of entities, which are sometimes called entity objects to clarify the distinction between EJB 2.1 entity beans and JPA entities (entity objects).

A JPA entity is a Java object that must match the following rules:

- ▶ It is a plain old Java object (POJO) that does not have to implement any particular interface or extend a special class.
- ▶ The class must not be declared final, and no methods or persistent instance variables must be declared final.
- ▶ The entity class must have a no-argument constructor that is public or protected. The entity class can have other constructors as well.
- ▶ The class must either be annotated with the `@Entity` annotation or specified in the `orm.xml` JPA descriptor. We use annotations in our examples.
- ▶ The class must define an attribute that is used to identify in an unambiguous way an instance of that class (it corresponds to the primary key in the mapped relational table).

- ▶ Both abstract and concrete classes can be entities, and entities can extend non-entity classes (this is a significant limitation with EJB 2.x).

**Note:** Entities have overcome the traditional limitation that is present in EJB 2.x entity beans: They can be transferred *on the wire* (for example, they can be serialized over RMI-IIOP). Of course, you must remember to implement the `java.io.Serializable` interface in the entity.

## A simple entity example

Example 5-1 shows a simple Customer entity with a few fields.

*Example 5-1 Simple entity class with annotations*

---

```
package itso.bank.entities;

@Entity

public class Customer implements java.io.Serializable {

    @Id
    private String ssn;

    private String title;
    private String firstname;
    private String lastname;

    public String getSsn() { return this.ssn; }
    public void setSsn(String ssn) { this.ssn = ssn; }

    // more getter and setter methods
}
```

---

- ▶ The **@Entity** annotation identifies a Java class as an entity.
- ▶ The **@Id** annotation is used to identify the property that corresponds to the primary key in the mapped table.
- ▶ The class is conforming to the JavaBean specification.

## 5.1.2 Mapping the table and columns

To specify the mapping of the entity to a database table, we use **@Table** and **@Column** annotations (Example 5-2).

*Example 5-2 Entity with mapping to a database table*

---

```
@Entity
@Table (schema="ITS0", name="CUSTOMER")
public class Customer implements java.io.Serializable {
```

```

@Id
@Column (name="SSN")
private String ssn;
@Column (name="NAME")
private String lastname;
private String title;
private String firstname;
.....

```

- ▶ The **@Table** annotation provides information related to which table and schema the entity corresponds to.
- ▶ The **@Column** annotation provides information related to which column is mapped by an entity property. By default, properties are mapped to columns with the same name, and the **@Column** annotation is used when the property and column names differ.

**Note:** Entities support two types of persistence mechanisms:

- ▶ Field-based persistence: The entity properties must be declared as public or protected and instruct the JPA provider to ignore getter/setters.
- ▶ Property-based persistence: You must provide getter/setter methods.

We recommend using the property-based approach, because it is more adherent to the Java programming guidelines.

The **@Column** annotation has further attributes that can be used to completely specify the mapping of an field/property (Table 5-1).

Table 5-1 @Column attributes

Attribute	Description
name	The name of the column
unique	True for UNIQUE columns
nullable	False for IS NOT NULL columns
insertable	True if the column is inserted on a create call
updatable	True if the column is updated when the field is modified
columnDefinition	SQL to create the column in a CREATE TABLE statement
table	Specified if the column is stored in a secondary table
length	Default length for a VARCHAR for CREATE TABLE

Attribute	Description
precision	Default length for a number definition for CREATE TABLE
scale	Default scale for a number definition for CREATE TABLE

The fields or properties supported by entities are:

- ▶ Java primitive types (byte, int, short, long, char).
- ▶ `java.lang.String`
- ▶ Other serializable types, including:
  - Wrappers of Java primitive types (Byte, Integer, Short, Long, Character)
  - `java.math.BigInteger`, `java.math.BigDecimal`
  - `java.util.Date`, `java.util.Calendar`, `java.sql.Date`
  - `java.sql.Time`, `java.sql.Timestamp`
  - `byte[]`, `Byte[]`, `char[]`, `Character[]`
  - User-defined serializable types
- ▶ Enumerated types
- ▶ Other entities and/or collections of entities, such as:
  - `java.util.Collection`
  - `java.util.Set`
  - `java.util.List`
  - `java.util.Map`
- ▶ Embeddable classes

**Note on inheritance:** Entities can extend another entity. In this case all the inherited properties are automatically persisted as well. This is not true if the entity extends a class that is not declared as an entity.

## Basic fields

The term *basic* is associated with a standard value persisted as-is to the data store.

```
@Basic
@Column(name="NAME")
private String lastname;
```

You can use the **@Basic** annotation on persistent fields of the following types:

- ▶ Primitives and primitive wrappers
- ▶ `java.lang.String`
- ▶ `byte[]`, `Byte[]`, `char[]`, `Character[]`
- ▶ `java.math.BigDecimal`, `java.math.BigInteger`

- ▶ `java.util.Date`, `java.util.Calendar`, `java.sql.Date`
- ▶ `java.sql.Timestamp`
- ▶ Enums
- ▶ Serializable types

This annotation is used very often when you want to specify a fetch policy.

### Transient fields

If you model a field inside an entity that you do not want to be persistent, you can use the `@Transient` annotation on the field/getter method (or simply use the transient modifier):

```
@Transient
private String fieldNotPersistent1;

transient String fieldNotPersistent2;
```

## 5.1.3 Entity identity

Every entity that is mapped to a relational database must have a mapping to a primary key in the table.

The JPA specification supports three different ways to specify the entity identity:

- ▶ `@Id` annotation
- ▶ `@IDClass` annotation
- ▶ `@EmbeddedId` annotation

### `@Id` annotation

The `@Id` annotation offers the simplest mechanism to define the mapping to the primary key.

You can associate the `@Id` annotation to fields/properties of these types:

- ▶ Primitive Java types and their wrapper classes
- ▶ Arrays of primitive or wrapper types
- ▶ Strings: `java.lang.String`
- ▶ Large numeric types: `java.math.BigDecimal`, `java.math.BigInteger`
- ▶ Temporal types: `java.util.Date`, `java.sql.Date`

We discourage the usage of floating point types (float and double, and their wrapper classes) for decimal data, because you can have rounding errors and the result of the equals operator is unreliable in these cases. Use `BigDecimal` instead. Temporal types can have similar problems due to time-based inconsistencies.

The `@Id` annotation fits very well in scenarios where a natural primary key is available, or when database designers use *surrogate primary keys* (typically an integer) that has no descriptive value and is not derived from any application data in the database.

On the other end, composite keys are useful when the primary key of the corresponding database table consists of more than one column. Composite keys can be defined by the `@IdClass` or `@EmbeddedId` annotation.

## **@IdClass**

In this section we illustrate how to model a composite key using the `@IdClass` annotation.

To map this composite key, follow these steps:

- ▶ Create a class that models the primary key, for example, `CustomerKey`:

```
public class CustomerKey implements Serializable {
    private int keyfield1;
    private String keyfield2;
    .....
}
```

  - The key class must implement the `java.io.Serializable` interface.
  - It overrides the `equals` and `hashCode` methods in order to use `organizationalUnit` and `userName` properties for the definition of Employee identity.
  - It has no setter methods, because once it has been constructed using the primary key values, it cannot be changed.
- ▶ Specify in the entity that we want to use the key class as our identity provider:

```
@IdClass(value=CustomerKey.class)
@Entity
public class Customer implements java.io.Serializable {
    @id
    private int keyfield1;
    @id
    private String keyfield2;
    .....
}
```

## **@EmbeddedId**

Using the `@EmbeddedId` annotation in conjunction with the `@Embeddable` annotation, the definition of the composite key is moved inside the entity:

```
@Entity
public class Customer implements Serializable {
```

```

    @EmbeddedId
    private CustomerKey key;
    .....
}
@Embeddable class CustomerKey {
    .....
}

```

The main feature of this refactoring are as follows:

- ▶ The key class does not have to implement `Serializable`.
- ▶ The key class is now nested (embedded) inside the entity class and therefore can be used only in conjunction with the entity.
- ▶ The entity does not redeclare the fields associated with the composite key.

**Note:** Generally speaking, the `@Embeddable` annotation is used to model persistent objects that have no identity of their own, because they are nested inside another entity.

## Entity automatic identity generation

Very often applications do not want to care explicitly about the identity of their managed entities, but instead require that identifier values be automatically generated for them. By default, the JPA persistence provider manages the provision of unique identifiers for entity primary keys using the `@Id` annotation.

This is where the `@GeneratedValue` annotation helps. JPA supports four different strategies for automatic identity generation (Table 5-2).

Table 5-2 identity generation strategies

Generation strategy	Description
<code>GenerationType.IDENTITY</code>	Specifies that the persistence provider uses a database identity column.
<code>GenerationType.SEQUENCE</code>	Specifies that the persistence provider use a database sequence (in conjunction with the <code>@SequenceGenerator</code> annotation).
<code>GenerationType.AUTO</code>	Specifies that the persistence provider should select a primary key generator that is most appropriate for the underlying database.

Generation strategy	Description
GenerationType.TABLE	Specifies that the persistence provider assigns primary keys for the entity using an underlying database table to ensure uniqueness (in conjunction with the @TableGenerator annotation).

### ***ID generation using database identity***

Some databases support a primary key identity column sometimes referred to as an autonumber column. In such cases, an identity column provides a way for these RDBMS to automatically generate a unique numeric value for each row in a table. A table can have a single column that is defined with the identity attribute.

### ***ID generation using database sequence***

A sequence is a database object that allows the automatic generation of numeric values.

While the identity column feature also allows automatic generation of numeric values, it is essentially an attribute of a column and thus exists for the period of existence of the table containing that column. On the other hand, a sequence exists outside of a table and is not tied to a column of a table. This allows more flexibility in using the sequence values in SQL operations.

### ***Automatic ID generation***

With this strategy, the persistence provider uses any strategy to generate identifiers.

### ***ID generation using a table generator***

Table generator technique uses a table in the database to generate unique IDs. The table has two columns, one stores the name of the sequence, the other stores the last ID value that was assigned.

There is a row in the sequence table for each sequence object. Each time a new ID is required the row for that sequence is incremented and the new ID value is passed back to the application to be assigned to an object.

**Note:** Table generator is the most portable solution because it just uses a regular database table, so unlike sequence and identity can be used on any RDBMS.

In JPA the @TableGenerator annotation is used to define a sequence table. The table generator defines pkColumnName for the column used to store the name of the sequence, valueColumnName for the column used to store the last ID



allocated, and `pkColumnValue` for the value to store in the name column (normally the sequence name).

### 5.1.4 Callback methods and listeners

JPA provides callback methods for performing actions at different stages of persistence operations. The callback methods can be annotated as follows:

- ▶ `@PostLoad`
- ▶ `@PrePersist`
- ▶ `@PostPersist`
- ▶ `@PreUpdate`
- ▶ `@PostUpdate`
- ▶ `@PreRemove`
- ▶ `@PostRemove`

For these annotations, the JSR 220 specification indicates:

- ▶ The `@PrePersist` and `@PreRemove` callback methods are invoked for a given entity before the respective persist and remove operations for that entity are executed by the persistence manager.
- ▶ The `@PostPersist` and `@PostRemove` callback methods are invoked for an entity after the entity has been made persistent or removed. These callbacks are also invoked on all entities to which these operations are cascaded. The `@PostPersist` and `@PostRemove` methods will be invoked after the database insert and delete operations respectively.
- ▶ The `@PreUpdate` and `@PostUpdate` callbacks occur before and after the database update operations to entity data. These database operations can occur at the time the entity state is updated, or they can occur at the time state is flushed to the database (which might be at the end of the transaction). Note that it is implementation-dependent as to whether `@PreUpdate` and `@PostUpdate` callbacks occur when an entity is persisted and subsequently modified in a single transaction, or when an entity is modified and subsequently removed within a single transaction. Portable applications should not rely on such behavior.
- ▶ The `@PostLoad` callback is invoked after the entity has been loaded into the current persistence context from the database or after the refresh operation has been applied to it. The `@PostLoad` method is invoked before a query result is returned or accessed or before an association is traversed.

These callback methods can be inserted in the entity class itself, or in a separate class and reference them in the entity class with a `@EntityListeners` annotation.

## 5.1.5 Relationships

Before starting our discussion of entity relationships, it is useful to refresh how the concept of relationships is defined in object-oriented and in relational database worlds (Table 5-3).

Table 5-3 Relationship concept in two different worlds

Java/JPA	RDBMS
A relationship is a reference from one object to another. Relationships are defined through object references (pointers) from a source object to the target object.	Relationships are defined through foreign keys.
If a relationship involves a collection of other objects, a collection or array type is used to hold the contents of the relationship.	Collections are either defined by the target objects have a foreign key back to the source object's primary key, or by having an intermediate join table to store the relationships.
Relationships are always unidirectional, in that if a source object references a target object, it is not guaranteed that the target object also has a relationship to the source object.	Relationships are defined through foreign keys and queries, such that the inverse query always exists.

JPA defines the following relationships: one-to-one, many-to-one, one-to-many, and many-to-many.

### One-to-one relationship

In this type of relationship each entity instance is related to a single instance of another entity. The `@OneToOne` annotation is used to define this single value association, for example, a `Customer` is related to a `CustomerRecord`:

```
@Entity
public class Customer {

    @OneToOne
    @JoinColumn(
        name="CUSTREC_ID", unique=true, nullable=false, updatable=false)
    public CustomerRecord getCustomerRecord() {
        return customerRecord;
    }
    ....
}
```

In many situations the target entity of the one-to-one has a relationship back to the source entity. In our example, `CustomerRecord` could have a reference back to the `Customer`. When this is the case, we call it a bidirectional one-to-one relationship.

There are two rules for bi-directional one-to-one associations:

- ▶ The `@JoinColumn` annotation must be specified in the entity that is mapped to the table containing the join column, or the owner of the relationship.
- ▶ The `mappedBy` element should be specified in the `@OneToOne` annotation in the entity that does not define a join column, that is, the inverse side of the relationship.

## Many-to-one and one-to-many relationships

Many-to-one mapping is used to represent simple pointer references between two Java objects. In Java, a single pointer stored in an attribute represents the mapping between the source and target objects. Relational database tables implement these mappings using foreign keys.

On the other side, one-to-many mapping is used to represent the relationship between a single source object and a collection of target objects. This relationship is usually represented in Java with a collection of target objects, but is more difficult to implement using relational databases (where you retrieve related rows through a query).

For example, an `Account` entity object can be associated with many `Transaction` entity objects:

```
@Entity
@Table (schema="ITS0", name="ACCOUNT")
public class Account implements Serializable {
    @Id
    private String id;
    private BigDecimal balance;

    @OneToOne(mappedBy="account")
    private Set<Transaction> transactionCollection;

    ....
}

@Entity
@Table (schema="ITS0", name="TRANSACTIONS")
public class Transaction implements Serializable {
    @Id
    private String id;
    .....
}
```

```

    @ManyToOne
    @JoinColumn(name="ACCOUNT_ID")
    private Account account;
    ....
}

```

### ***Using the @JoinColumn annotation***

In the database, a relationship mapping means that one table has a reference to another table. The database term for a column that refers to a key (usually the primary key) in another table is a foreign key column.

In the Java Persistence API we call them join columns, and the `@JoinColumn` annotation is used to configure these types of columns.

**Note:** If you do not specify `@JoinColumn`, then a default column name is assumed. The algorithm used to build the name is based on a combination of both the source and target entities. It is the name of the relationship attribute in the Transaction source entity (the **account** attribute), plus an underscore character (`_`), plus the name of the primary key column of the target Account entity (the **id** attribute).

Therefore a foreign key named **ACCOUNT\_ID** is expected inside the **TRANSACTION** table. If this is not applicable, you must use `@JoinColumn` to override this automatic behavior.

The `@JoinColumn` annotation also applies to one-to-one relationships.

## **Many-to-many relationship**

When an entity A references multiple B entities, and other As might reference some of the same Bs, we say there is a many-to-many relation between A and B. To implement a many-to-many relationship there must be a distinct join table that maps the many-to-many relationship. This is called an association table.

For example, a Customer entity object can be associated with many Account entity objects, and an Account entity object can be associated with many Customer entity objects using the `@ManyToMany` annotation:

```

@Entity
public class Customer implements Serializable {
    .....

    @ManyToMany
    @JoinTable(name="ACCOUNTS_CUSTOMERS", schema="ITS0",
        joinColumns=@JoinColumn(name="CUSTOMERS_SSN"),
        inverseJoinColumns=@JoinColumn(name="ACCOUNTS_ID") )

```

```

        private Set<Account> accountCollection;
        .....
    }

    @Entity
    public class Account implements Serializable {
        .....

        @ManyToMany(mappedBy="accountCollection")
        private Set<Customer> customerCollection;
        ....
    }

```

The `@JoinTable` annotation is used to specify a table in the database that associates customers with accounts. The entity that specifies the `@JoinTable` is the owner of the relationship, so in this case the `Customer` entity is the owner of the relationship with the `Account` entity.

The join column pointing to the owning side is described in the `joinColumns` element, while the join column pointing to the inverse side is specified by the `inverseJoinColumns` element.

**Note:** Neither the `CUSTOMER` nor the `ACCOUNT` table contains a foreign key. The foreign keys are in the association table. Therefore, the `Customer` or the `Account` entity can be defined as the owning entity.

## Fetch modes

When an entity manager retrieves an entity from the underlying database, it can use two types of strategies:

- ▶ **Eager mode:** When you retrieve an entity from the entity manager or by using a query, you are guaranteed that all of its fields (with relationships too) are populated with data store data.
- ▶ **Lazy mode:** This is a hint to the JPA runtime that you want to defer loading of the field until you access it. Lazy loading is completely transparent, when you attempt to read the field for the first time, the JPA runtime will load the value from the data store and populate the field automatically:

```

    @OneToOne(mappedBy="accounts", fetch=FetchType.LAZY)
    private Set<Transaction> transactionCollection;

```

**Note:** The use of eager mode can greatly impact the performance of an application, especially if entities have many and recursive relationships, because all the entity will be loaded at once.

On the other hand, if the entity after that has been read by the entity manager, is detached and sent over the network to another layer, you usually should assure that all the entity attributes have been read from the underlying data store, or that the receiver does not require related entities.

The default value of the fetch strategy changes according to where it is used (Table 5-4).

Table 5-4 Default fetch strategies

Context	Default fetch mode
@Basic	EAGER
@OneToMany	LAZY
@ManyToOne	EAGER
@ManyToMany	LAZY

## Cascade types

When the entity manager is reading, updating, or deleting an entity, you can instruct it to automatically cascade the operation to the entities held in a persistent field with the *cascade* property of the metadata annotation. This process is recursive:

```
@OneToMany(mappedBy="accounts",
            cascade={CascadeType.PERSIST,CascadeType.REMOVE})
private Set<Transaction> transactionCollection;
```

The cascade property accepts an array of CascadeType enum values (Table 5-5).

Table 5-5 Cascade rules

Cascade rule	Description
CascadeType.PERSIST	When persisting an entity, also persist the entities held in this field. This rule should always been used if you want to guarantee all relationships linked to be automatically persisted when a entity field/attribute is modified.
CascadeType.REMOVE	When deleting an entity, also delete the entities held in this field.

Cascade rule	Description
CascadeType.REFRESH	When refreshing an entity, also refresh the entities held in this field.
CascadeType.MERGE	When merging entity state, also merge the entities held in this field.
CascadeType.ALL	This rule acts as a shortcut for all of the foregoing values.

## 5.1.6 Entity inheritance

For several years we heard the term *impedance mismatch*, which describes the difficulties in bridging the object and relational worlds. Unfortunately, there is no natural and efficient way to represent an inheritance relationship in a relational database.

JPA introduces three strategies to support inheritance:

- ▶ Single table
- ▶ Joined tables
- ▶ Table per class

### Single table inheritance

This strategy maps all classes in the hierarchy to the base class table. This means that the table contains the superset of all the data contained in the class hierarchy.

The main advantage of this strategy is that is the fastest of all inheritance models, because it never requires a join to retrieve a persistent instance from the database. Similarly, persisting or updating a persistent instance requires only a single INSERT or UPDATE statement.

However, if the hierarchy model becomes wide or deep, the mapped table must have columns for every field in the entire inheritance hierarchy, which results in designing tables with nullable columns, that will be mostly empty.

For example, if the `Transaction` entity has two subclasses `Credit` and `Debit`, and the underlying table has a *discriminator* column named `transtype` to distinguish between the subclasses, then we specify single table inheritance using the `@Inheritance`, `@DiscriminatorColumn`, and `@DiscriminatorValue` annotations:

```
@Entity
@Table (schema="ITS0", name="TRANSACTIONS")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="transtype",
```

```

        discriminatorType=DiscriminatorType.STRING, length=32)
public abstract class Transaction implements Serializable {
    ...
}

@Entity
@DiscriminatorValue("Debit")
public class Debit extends Transaction {
    ...
}

@Entity
@DiscriminatorValue("Credit")
public class Credit extends Transaction {
    ...
}

```

## Joined tables inheritance

With this strategy, the top level entry in the entity hierarchy is mapped to a table that contains columns common to all the entities, while each of the other entities down the hierarchy are mapped to a table that contain only columns specific to that entity.

To reassemble an instance of any of the subclasses, the tables of the subclasses must be joined together with the superclass tables. That is why this strategy is called the joined strategy.

For each subclass, we use the `@Table` annotation to specify the joined table, the `@DiscriminatorValue` annotation to specify the value in the discriminator column, and the `@PrimaryKeyJoinColumn` annotation to specify the foreign key column:

```

@Entity
@Table(schema="ITS0", name="CREDIT")
@DiscriminatorValue(value="Credit")
@PrimaryKeyJoinColumn(name="TRANS_ID")
public class Credit extends Transaction {
    ...
}

```

## Table per class inheritance

With this strategy both the superclass and subclasses are stored in their own table and no relationship exists between any of the tables. Therefore, all the entity data are stored in their own tables.

Because every entity is mapped to a separate table, you must specify the `@Table` annotation for each of them.



The table-per-class strategy is very efficient when operating on instances of a known class, because you do not have to join to superclass or subclass tables. Reads, joins, inserts, updates, and deletes are all efficient in the absence of polymorphic behavior. Also, as in the joined strategy, adding additional classes to the hierarchy does not require modifying existing class tables.

However, with this strategy, when the concrete subclass is not known, and therefore the related object could be in any of the subclass tables, you cannot use joins, because there are no relationships. This issue involves identity lookups and queries too, and these operations require multiple SQL SELECTs (one for each possible subclass), or a complex UNION, and therefore you might have performance problems as well.

Because of these drawbacks, the table per class inheritance implementation is not used much in practice.

### 5.1.7 Persistence units

A persistence unit defines a set of all entity classes that are managed by entity manager (described hereafter) instances in an application. This set of entity classes represents the data contained within a single data store.

Persistence units are defined by the `persistence.xml` configuration file. The JAR file or directory whose `META-INF` directory contains `persistence.xml` is called the root of the persistence unit. The scope of the persistence unit is determined by the persistence unit's root.

Example 5-3 shows an extract of the `persistence.xml` file.

*Example 5-3 Extract of a persistence.xml file*

---

```
<persistence version="1.0" .....>
  <persistence-unit name="EJB3Bank" transaction-type="JTA">
    <jta-data-source>jdbc/ejb3bank</jta-data-source>
    <non-jta-data-source>jdbc/itsoejb30nojta</non-jta-data-source>
    <class>itso.bank.entities.Account</class>
    <class>itso.bank.entities.Customer</class>
    <class>itso.bank.entities.Transactions</class>
  </persistence-unit>
</persistence>
```

---

Each persistence unit must be identified with a name that is unique to the persistence unit's scope. Persistent units can be packaged as part of a WAR or EJB JAR file, or can be packaged as a JAR file that can then be included in a WAR or EAR file. If you package the persistent unit as a set of classes in an EJB JAR, the `persistence.xml` file should be put in the EJB JAR's `META-INF` directory.

In Java EE, the root of a persistence unit can be one of the following items:

- ▶ An EJB JAR file
- ▶ The WEB-INF/classes directory of a WAR file
- ▶ A JAR file in the WEB-INF/lib directory of a WAR file
- ▶ A JAR file in the root of the EAR
- ▶ A JAR file in the EAR library directory
- ▶ An application client JAR file

Table 5-6 illustrates the meaning of the main elements of this file.

*Table 5-6 Main elements in the persistence.xml file*

Element	Description
name	Identifies the persistence unit and is used to refer the persistence unit with annotation injection or deployment descriptors. The name must be unique within the scope of the persistence unit packaging.
provider	Specifies the persistence provider; this element is optional and usually is not specified, so that the default implementation provided by the Java EE application server is used.
transaction-type	<p>Specifies if the persistence unit will participate in a JTA transaction managed by the container or it will managed by the EntityTransaction interface. It assumes two values:</p> <ul style="list-style-type: none"><li>▶ JTA</li><li>▶ RESOURCE_LOCAL</li></ul> <p>Usually you do not have to specify this element, because it is automatically detected by the runtime environment. The default value is based on whether the application is using container-managed entity managers (JTA) or application-managed entity managers (RESOURCE_LOCAL). So, if you are moving from a unit-test development environment to a more production environment, these default values can come into play.</p>
jta-data-source	(optional) Identifies the global JNDI name of the data source that can be used in JTA transactions.
non-jta-data-source	(optional) Non-JTA-enabled data source to be used by the persistence provider for accessing data outside a JTA. These non-jta data sources are the defined mechanisms for accessing the database for ID generation (table and sequence).
class	Identifies the list of Entities defined inside this persistence unit; if this attribute is not specified, the EJB container automatically detects all the entities marked with @Entity annotation.

Element	Description
mapping-file	(optional) Name of an XML mapping file containing persistence information to be included in this persistence unit.
properties	(optional) Defines the set of vendor-specific attributes passed to the persistence provider.

**Note:** The persistence.xml file is mandatory and cannot be substituted with the use of annotations.

### 5.1.8 Object-relational mapping through orm.xml

As we have seen in this chapter, the object-relational (o/r) mapping of an entity can be done through the use of annotations. As an alternative, you can specify the same information in an external file (called **orm.xml**) that must be packaged in the META-INF directory of the persistence module, or in a separate file packaged as a resource and defined in persistence.xml with the mapping-file element.

Example 5-4 shows an extract of an orm.xml file that maps the Account entity to the ITS0.ACCOUNT table.

*Example 5-4 Extract of an orm.xml file to define an entity mapping*

```
<entity-mappings .....>
  <entity class="itso.bank.entity.Account" metadata-complete="true"
    name="Account">
    <description>Account of ITS0 Bank</description>
    <table name="ACCOUNT" schema="ITS0"></table>
    <attributes>
      <id name="accountNumber"><column name="id"/></id>
      <basic name="balance"></basic>
      <one-to-many name="transactionCollection"></one-to-many>
    </attributes>
  </entity>
  .....
</entity-mappings>
```

When using the orm.xml file with the metadata-complete="true" value, no annotations should be present in the entity classes and the complete mapping must be specified in the orm.xml file. For more information about the content of the orm.xml file, refer to the JPA specification.

## 5.1.9 Persistence provider

Persistence providers are implementations of the Java Persistence API (JPA) specification and can be deployed in the Java EE compliant application server that supports JPA persistence.

WebSphere Application Server includes two persistence providers:

- ▶ The Apache OpenJPA persistence provider
- ▶ The JPA for WebSphere Application Server persistence provider

## 5.1.10 Entity manager

Entities cannot persist themselves on the relational database; annotations are used only to declare a POJO as an entity or to define its mapping and relationships with the corresponding tables on the relational database.

JPA has defined the `EntityManager` interface for this purpose to let applications manage and search for entities in the relational database.

The `EntityManager` primary definition is:

- ▶ An API that manages the life cycle of entity instances.
- ▶ Each `EntityManager` instance is associated with a *persistence context*.
- ▶ A persistence context defines the scope under which particular entity instances are created, persisted, and removed through the APIs made available by an `EntityManager`.
- ▶ An object that manages a set of entities defined by a persistence unit.

The entity manager tracks all entity objects within a persistence context for changes and updates made, and flushes these changes to the database. After a persistence context is closed, all managed entity object instances become detached from the persistence context and its associated entity manager, and are no longer managed.

**Managed and unmanaged entities:** An entity object instance is either *managed* (attached) by an entity manager or *unmanaged* (detached):

- ▶ When an entity is attached to an entity manager, the manager monitors any changes to the entity and synchronizes them with the database whenever the entity manager decides to flush its state.
- ▶ When an entity is detached, and therefore is no more associated with a persistence context, it is unmanaged, and its state changes are not tracked by the entity manager and synchronized with the database.

The main operations that can be performed by an entity manager are shown in Table 5-7.

Table 5-7 *PersistenceManager API*

Operation (method)	Description
<code>persist</code>	<ul style="list-style-type: none"><li>▶ Insert a new entity instance into the database.</li><li>▶ Save the persistent state of the entity and any owned relationship references.</li><li>▶ Entity instance becomes managed.</li></ul>
<code>find</code>	Obtain a managed entity instance with a given persistent identity (primary key), return null if not found.
<code>remove</code>	Delete a managed entity with the given persistent identity from the database.
<code>merge</code>	<ul style="list-style-type: none"><li>▶ State of detached entity gets merged into a managed copy of the detached entity.</li><li>▶ Managed entity that is returned has a different Java identity than the detached entity.</li></ul>
<code>refresh</code>	Reload the entity state from the database.
<code>lock</code>	Set the lock mode for an entity object contained in the persistence context.
<code>flush</code>	Force synchronization with database.
<code>contains</code>	Determine if an entity is contained by the current persistence context.
<code>createQuery</code>	Create a query instance using dynamic Java Persistent Query Language.
<code>createNamedQuery</code>	Create instance of a predefined query.
<code>createNativeQuery</code>	Create instance of an SQL query.

Two types of entity manager are available:

- ▶ Container-managed entity manager
- ▶ Application-managed entity manager

### Container-managed entity manager

The most common and widely used entity manager in a Java EE environment is the container-managed entity manager. In this mode, the container is responsible for the opening and closing of the entity manager and thus, the life cycle of the persistence context (this is transparent to the application).

A container-managed entity manager is also responsible for transaction boundaries. A container-managed entity manager is obtained in an application through dependency injection or through JNDI lookup, and the container manages interaction with the entity manager factory transparently to the application.

A container-managed entity manager requires the use of a JTA transaction, because its persistence context will automatically be propagated with the current JTA transaction, and the entity manager references that are mapped to the same persistence unit will provide access to this same persistence context within the JTA transaction.

This propagation of persistence context by the Java EE container avoids the need for the application to pass entity manager references from one component to another.

Container-managed persistence contexts can be defined to have one of two different scopes:

- ▶ Transaction persistence scope
- ▶ Extended persistence scope

### ***Transaction persistence scope***

Within this scope, contexts live as long as a transaction and are closed when the transaction completes. When the transaction-scoped persistence context is destroyed, all managed entity object instances become detached. Transaction scoped persistence is the most common within a container.

**Note:** Entity manager instances are not thread safe. This means that you cannot inject it inside a servlet. You must instead use JNDI lookup and assign the result to a local variable (not an instance variable), otherwise you could have unpredictable results in your Java EE application.

### ***Extended persistence context***

Persistence contexts can be configured to live longer than a transaction. This is called an *extended persistence* context. Entity object instances that are attached to an extended context remain managed, and attached to the associated entity manager, even after a transaction is complete.

A container-managed persistence context that has an extended life time begins when a stateful session bean is created and ends when the stateful session bean is removed from the container.

**Note:** Only stateful session beans can have a container-managed, extended persistence context.

One of the important distinctions between a transaction-scoped persistence context and that of an extended persistence context is the state of the entities after a transaction completes:

- ▶ In the case of a transaction-scoped persistence context, the entities become detached, that is, they are no longer managed.
- ▶ In the case of an extended persistence context, the entities remain managed. In other words, all operations, such as persist, that are done outside of a transaction are queued and committed when the persistence context is attached to a transaction (and when it commits). This scenario is used for all the workflows/use cases that span multiple transactions.

### **Application-managed entity manager**

An application-managed entity manager allows you to control the entity manager in application code. When using such an entity manager, you should be aware of the following fundamental aspects:

- ▶ With application-managed entity managers the persistence context is not propagated to application components, and the life cycle of entity manager instances is managed by the application.
- ▶ This means that the persistence context is not propagated with the JTA transaction across entity manager instances in a particular persistence unit.
- ▶ Furthermore, the entity manager, and its associated persistence context, is created and destroyed explicitly by the application.

Taking into account these peculiarities, this entity manager is usually used in two different scenarios:

- ▶ In Java SE environments, where you want to access a persistence context that is stand-alone, and not propagated along with the JTA transaction across the entity manager references for the given persistence unit.
- ▶ Inside a Java EE container, when you want to gain a very fine-grained control over the entity manager life cycle.

### ***Application-managed entity manager in a Java SE scenario***

If you want to use JPA outside a Java EE container, you must use resource-local transactions (because JTA is not available) through the APIs provided by the `EntityManager` interface.

### 5.1.11 Entity life cycle

An entity manager instance is associated with a persistence context. Within this persistence context, the entity instances and their life cycle are managed and can be accessed through the operations described in Table 5-7 on page 145.

Entity instances become unmanaged and detached when a transaction scope or extended persistence context ends. A very important consequence is that detached entities can be serialized and sent across the network to a remote client. The client can make changes remotely to these serialized object instances and send them back to the server to be merged back and synchronized with the database.

**Note:** This behavior is very different from the EJB 2.1 entity model, where entities are always managed by the container. In EJB 3.0 you must always remember that you are working with entities that are POJOs.

This can be considered as a real innovative (and, above all, simplified) model in designing Java EE applications, because you are not forced any more to use patterns, such as data transfer objects (DTO), between the business logic layer (session beans) and the persistence layer.

Whether to propagate this new approach to clients of an EJB 3.0 business layer and (for instance a Web MVC controller or a Rich Eclipse Platform application) has been long debated, both inside the communities and in literature as well.

Here (without going into too deep details), we can remark that before the coming of EJB 3.0, two fundamental approaches have been used to model Java EE applications:

- ▶ EJB facade pattern, where session beans are used as facades and coordinate the access to the back-end enterprise information system.
- ▶ POJO facade, where the facade pattern is directly implemented by POJOs and transactional and remote services are provided by frameworks such as Spring.

These are two models that were sharply separated, but find their natural convergence in EJB 3.0, because session beans are POJOs as well.

### 5.1.12 JPA query language

The Java persistence query language (JPQL) is used to define searches against persistent entities independent of the mechanism used to store those entities. As such, JPQL is *portable*, and not constrained to any particular data store.



The Java persistence query language is an extension of the Enterprise JavaBeans query language, EJB QL, and is designed to combine the syntax and simple query semantics of SQL with the expressiveness of an object-oriented expression language:

- ▶ The application creates an instance of the `javax.persistence.EntityManager` interface.
- ▶ The `EntityManager` creates an instance of the `javax.persistence.Query` interface, through its public methods, for example, `createNamedQuery`.
- ▶ The `Query` instance executes a query (to read or update entities).

## Query types

Query instances are created using the methods exposed by the `EntityManager` interface (Table 5-8).

*Table 5-8 How to create a Query instance*

Method name	Description
<code>createQuery(String qlString)</code>	Create an instance of <code>Query</code> for executing a Java Persistence query language statement.
<code>createNamedQuery (String name)</code>	Create an instance of <code>Query</code> for executing a named query (in the Java Persistence query language or in native SQL).
<code>createNativeQuery (String sqlString)</code>	Create an instance of <code>Query</code> for executing a native SQL statement, for example, for update or delete.
<code>createNativeQuery (String sqlString, Class resultClass)</code>	Create an instance of <code>Query</code> for executing a native SQL query that retrieves a single entity type.
<code>createNativeQuery (String sqlString, String resultSetMapping)</code>	Create an instance of <code>Query</code> for executing a native SQL query statement that retrieves a result set with multiple entity instances.

## Query basics

A simple query that retrieves all the `Customer` entities from the database is shown here:

```
EntityManager em = ...
Query q = em.createQuery("SELECT c FROM Customer c");
List<Customer> results = (List<Customer>)q.getResultList();
```

A JPQL query has an internal name space declared in the from clause of the query. Arbitrary identifiers are assigned to entities so that they can be referenced elsewhere in the query. In the previous query example, the identifier *c* is assigned to the Customer entity.

The where condition is used to express a logical condition:

```
EntityManager em = ...
Query q = em.createQuery("SELECT c FROM Customer c
where c.ssn='111-11-1111'");
List<Customer> results = (List<Customer>)q.getResultList();
```

## Operators

JPQL provides several operators; the most important are:

- ▶ Logical operators: NOT, AND, OR
- ▶ Relational operators: =, >, >=, <, <=, <>, [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL, IS [NOT] EMPTY, [NOT] MEMBER [OF]
- ▶ Arithmetic operators: +, -, /, \*

## Named queries

JPQL defines two types of queries:

- ▶ **Dynamic queries:** These queries are created on the fly.
- ▶ **Named queries:** These queries are intended to be used in contexts where the same query is invoked several times. Their main benefits include the improved reusability of the code, a minor maintenance effort, and finally better performance, because they are evaluated once.

**Note:** From this point of view, there is a strong similarity between dynamic/named queries and JDBC Statement/PreparedStatement. However, named queries are stored in a global scope, which enables them to be accessed by different EJB 3.0 components.

### *Defining a named query*

Named queries are defined using the **@NamedQuery** annotation:

```
@Entity
@Table (schema="ITS0", name="CUSTOMER")
@NamedQuery (name="getCustomerBySSN",
            query="select c from Customer c where c.ssn = ?1")
public class Customer implements Serializable {
    ...
}
```

The name attribute is used to uniquely identify the named query, while the query attribute defines the query. We can see how this syntax resembles the syntax used in JDBC code with `jdbc.sql.PreparedStatement` statements.

Instead of a positional parameter (`?1`), the same named query can be expressed using a named parameter:

```
@NamedQuery(name="getCustomerBySSN",
            query="select c from Customer c where c.ssn = :ssn")
```

### ***Completing a named query***

Named queries must have all their parameters specified before being executed. The `javax.persistence.Query` interface exposes two methods:

```
public void setParameter(int position, Object value)
public void setParameter(String paramName, Object value)
```

A complete example that uses a named query is shown here:

```
EntityManager em = ...
Query q = em.createNamedQuery("getCustomerBySSN");
q.setParameter(1, "111-11-1111");
//q.setParameter("ssn", "111-11-1111"); // for named parameter
List<Customer> results = (List<Customer>)q.getResultList();
```

### **Defining multiple named queries**

If there are more than one named query for an entity, they are placed inside an **@NamedQueries** annotation, which accepts an array of one or more **@NamedQuery** annotations:

```
@NamedQueries({
    @NamedQuery(name="getCustomers",
        query="select c from Customer c"),
    @NamedQuery(name="getCustomerBySSN",
        query="select c from Customer c where c.ssn =?1"),
    @NamedQuery(name="getAccountsBySSN",
        query="select a from Customer c, in(c.accountCollection) a
            where c.ssn =?1 order by a.accountNumber")
})
```

### **Updating and deleting instances**

Of course, JPQL can be used to update or delete entities.

### **Retrieving a single entity**

If you want to retrieve a single instance, the `Query` interface provides the `getSingleResult` method. This method has some significant characteristics:

- It throws a `NoResultException` if there is no result.

- ▶ It throws a `NonUniqueResultException` if more than one result.
- ▶ It throws an `IllegalStateException` if called for a Java Persistence query language UPDATE or DELETE statement.

**Note:** Even if these three exceptions are unchecked, they do not cause the provider to roll back the current transaction. These exceptions must be managed by the application code.

## Relationship navigation

Relations between objects can be traversed using Java-like syntax:

```
SELECT t FROM Transaction t WHERE t.account.id = '001-111001'
```

There are other ways to use queries to traverse relationships. For example, if the `Account` entity has a property called `transacationCollection` that is annotated as a `@OneToMany` relationship, then this query retrieves all the `Transaction` instances of one `Account`:

```
@NamedQuery(name="getTransactionsByID",
            query="select t from Account a, in(a.transactionsCollection) t
                  where a.id =?1 order by t.transtime")
```

## Query paging

Query paging is a very common design pattern in applications that work with a corporate database, which generally contains a huge amount of data. The design of these applications must take into account not to generate queries that could read a very large number of records, because this could easily consume the necessary limited physical resources of the operating environment (RDBMS and Java EE application servers first).

The `Query` interface has two specific methods to limit the number of returned entities:

- ▶ Set the maximum number of results to retrieve:

```
setMaxResults(int maxResults)
```

- ▶ Set the position of the first result to retrieve:

```
setFirstPosition (int firstPosition)
```

## Flush mode and queries

When queries retrieve data inside a transaction, the returned entities can be deeply impacted by the pending operations that are currently ongoing in the same transaction (within the same persistence context).

The Query interface supports a specific method `setFlushMode (int flushMode)` to fine control the flushing of the persistence context before a specific query is executed. The `flushMode` can assume two values:

- ▶ `FlushModeType.AUTO` (default value): The persistence provider is responsible for ensuring that all updates to the state of all entities in the persistence context, which could potentially affect the result of the query, are visible to the processing of the query. The persistence provider implementation can achieve this by flushing those entities to the database or by some other means.
- ▶ `FlushModeType.COMMIT`: The effect of updates made to entities in the persistence context upon queries is not detailed by the JPA specification, and depends on how the persistence provider implements the query integrity support.<sup>1</sup>

### Polymorphic queries

JPQL queries are polymorphic, which means that the from clause of a query includes not only instances of the specific entity class to which it refers, but all subclasses of that class as well. The instances returned by a query include instances of the subclasses that satisfy the query conditions.

## 5.2 Jump start

For most chapters of this book, you can jump start to a chapter by importing the code of the previous chapters as a ZIP file. However, the jump start concept does not apply for this chapter, because it is the first chapter in the Donate application scenario. By definition, you must complete the preceding chapters before starting this chapter:

- ▶ Chapter 2, “Install and configure software” on page 25
- ▶ Chapter 3, “Configure the development environment” on page 53
- ▶ Chapter 4, “Prepare the legacy application” on page 85

## 5.3 Create the database connection

To create JPA entities from an existing database, we must have the database and tables defined. The `Vacation` database was created along with a workbench data source connection in 4.4, “Create and configure the Vacation database” on page 90.

---

<sup>1</sup> OpenJPA and TopLink update the database at commit regardless of the `FlushModeType` setting, whereas Hibernate saves changes before a query is executed.


## 5.4 Create an enterprise application project

In this section, we create the DonateEAR enterprise application project that we use in this scenario:

1. In the workbench action bar, select **File** → **New** → **Enterprise Application Project**.

### Alternatives:

You can also create the Enterprise Application Project through several other methods.

- ▶ In the Enterprise Explorer, select in the white space, right-click and select **New** → **Enterprise Application Project**.
- ▶ In the Java EE perspective workbench action bar, click the **New** icon () and select **Enterprise Application Project** from the pull-down.

2. At the New EAR Application Project/EAR Application Project pop-up (Figure 5-2 on page 155) Set the following values:
  - Set Project name to **DonateEAR**.
  - Set Target Runtime to **WebSphere Application Server v7.0**
  - Set EAR version to **5.0**.
  - Set Configuration to **Default Configuration for WebSphere Application Server v7.0**.
- a. Click **Next**. Note that the navigation buttons are typically at the bottom of the pop-ups, and that in this book they are often cropped out to minimize the additional whitespace and pages that would result from larger images.

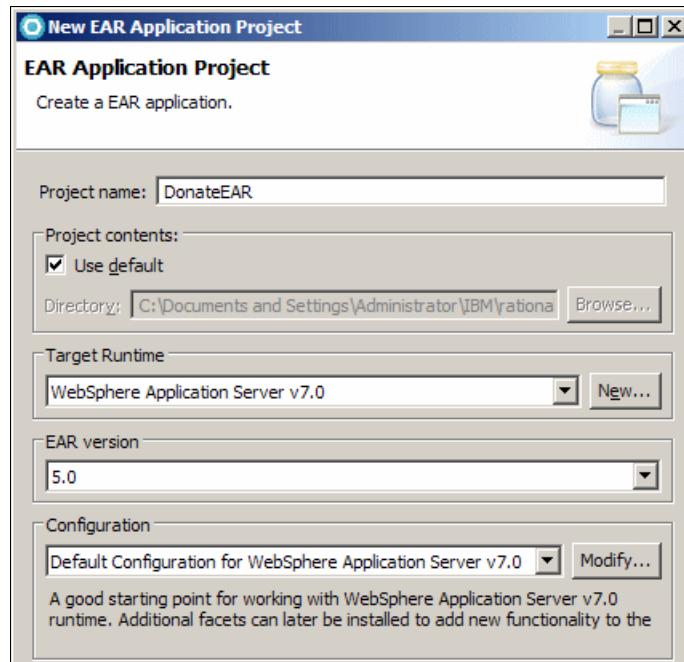


Figure 5-2 EAR application for DonateEAR

### Behind the scenes:

- ▶ EAR version refers to the Java EE (J2EE) version. WebSphere Application server V7.0 supports both J2EE 1.4 (1.4) and Java EE 5 (5.0), and in this book you are using Java EE 5.
- ▶ The Configuration setting allows you to override the specific functions that capabilities that are valid for the combination of this project and the target runtime. A user could create a custom configuration that provides a reduced set of facets (functions) or an extended set.

Project facets are a feature of Eclipse that allows the developer to include optional characteristics and requirements in a project. For example, in this case DonateEAR is defined as being an enterprise application (EAR) that runs on WebSphere Application server V7.0, supporting base Java EE capabilities (co-existence) and WebSphere Extensions.

You can change the list of project facets after a project is created by opening the project properties (select the project, right-click and select **Properties**) and adding or removing facets on the Project Facets tab.

3. At the New EAR Application Project/Enterprise Application pop-up (Figure 5-3), select **Generate Deployment Descriptor** and click **Finish**.

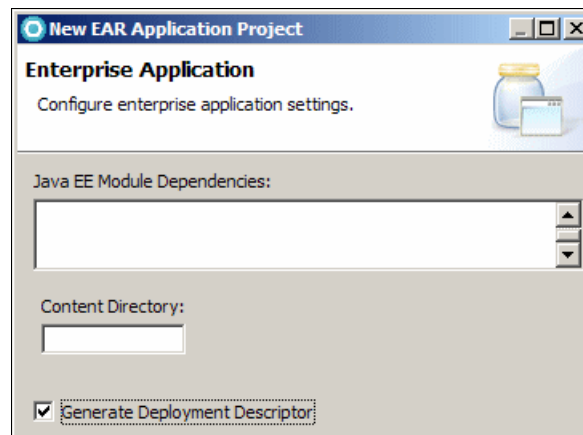


Figure 5-3 Generate deployment descriptor

4. DonateEAR now exists in the Enterprise Explorer (Figure 5-4).



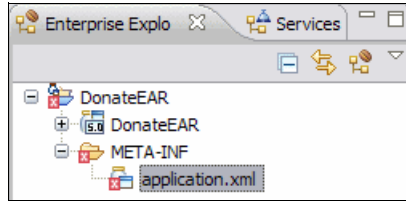



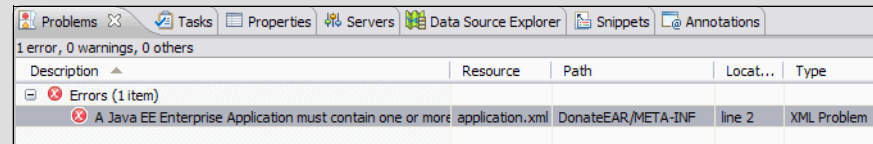
Figure 5-4 Enterprise Explorer with DonateEAR

### Behind the scenes:

You receive an error in the `application.xml` file at this point because we have not added any projects to the EAR yet. The Java EE specification requires an Enterprise Application to contain at least one module.

This error will be resolved when you add the `DonateEJB` project to this EAR in 6.3.1, “Create the Project” on page 221.

- ▶ The error is visible in the Enterprise Explorer through red error flags [  ] in the project tree hierarchy.
- ▶ The error is listed in the Problem view (tab) in the lower right pane.



## 5.5 Create and configure a JPA project

In this section, we create the `DonateJPAEmployee` project to be contained in the EAR project. A JPA project generally contains JPA entities, which are packaged as a JAR file. These JAR files are re-packaged in an EAR file for deployment.

In our case, the JPA project will contain the `Employee` entity that maps to the `VACATION.EMPLOYEE` table. We use reverse engineering to create the `Employee` entity from the database table.

### 5.5.1 Create the `DonateJPAEmployee` project

Follow these steps to create the project:

1. In the Enterprise Explorer, select anywhere in the white space, right-click and select **New** → **Other**.

2. At the New pop-up, select **JPA** → **JPA Project** and click **Next**.
3. At the New JPA Project/JPA Project pop-up (Figure 5-5 on page 158):
  - Set the project name to **DonateJPAEmployee**.
  - Set the configuration to **Default Configuration for WebSphere Application Server v7.0**

#### Behind the scenes:

The Default Configuration for WebSphere Application Server v7.0 project facet uses Java 6.0. A project facet is a set of configurations that can be added to a project. If your project is using Java version 5.0 use the project facet *Utility JPA project with Java 5.0*

- a. Select **Add project to an EAR**.
- b. Set Ear Project Name to **DonateEAR**.
- c. Click **Next**.

**JPA Project**  
Configure JPA project settings.

Project name:

Project contents:  
☒ Use default  
Directory:

Target Runtime

Configuration  
   
A good starting point for working with WebSphere Application Server v7.0 runtime. Additional facets can later be installed to add new functionality to the

EAR Membership  
☒ Add project to an EAR  
EAR Project Name:

Figure 5-5 New JPA Project/JPA Project pop-up DonateJPAEmployee

4. At the New JPA Project/JPA Facet pop-up (Figure 5-6 on page 160):
  - Set Platform to **RAD JPA Platform**.
  - Set Connection to **Vacation**.

**Off course:**

If you receive a message indicating no connection, follow the steps in 4.4.1, “Create the Vacation Derby database” on page 90

If you cannot connect, then make sure you start the Derby network sever as described in 4.2, “Start the Derby Network Server” on page 87

- a. Select **Use implementation provided by server runtime**.
- b. Select **Discover annotated classes automatically**.
- c. Select **Create orm.xml**.
- d. Click **Finish**.

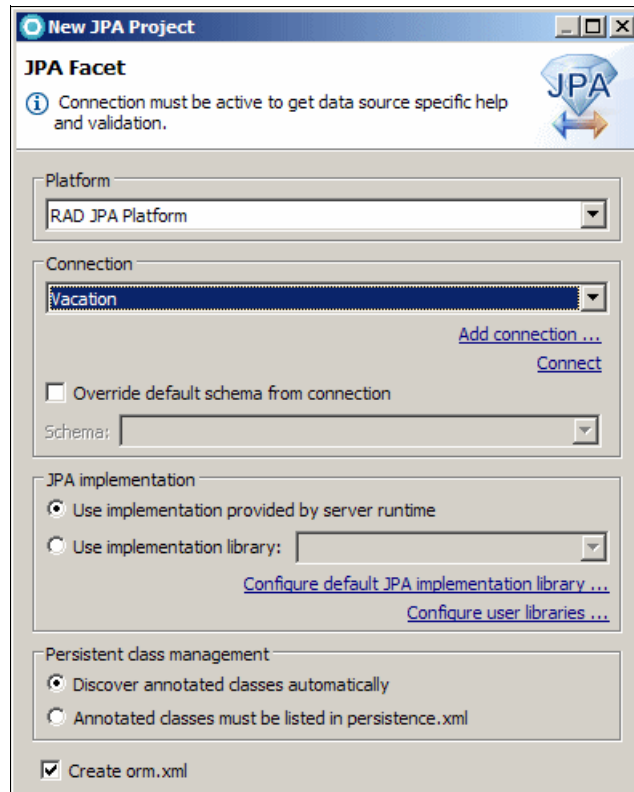


Figure 5-6 New JPA Project set RAD JPA platform

5. When prompted, switch to the JPA perspective(Figure 5-7 on page 160).

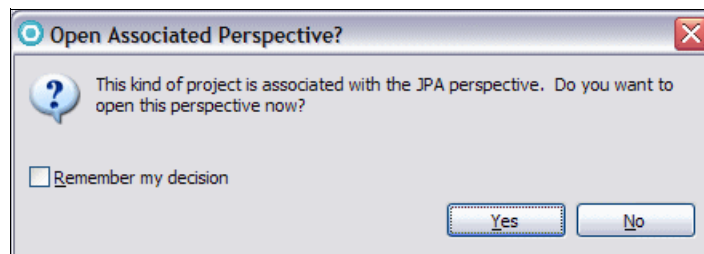


Figure 5-7 Open Associated Perspective message

**Behind the scenes:** The JPA perspective provides special views for JPA development, such as JPA Structure (top right) and JPA Details (bottom right).

6. Verify that the `DonateJPAEmployee` project is visible in the Project Explorer. Notice the `persistence.xml` and `orm.xml` files under `src/META-INF` (Figure 5-8).

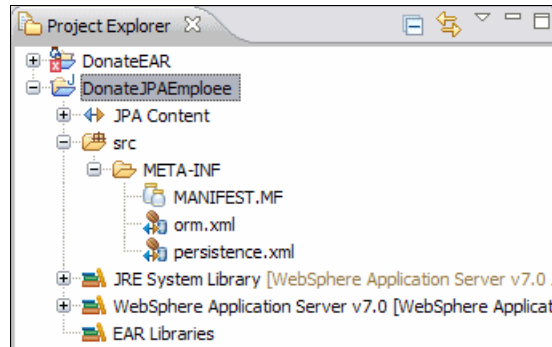


Figure 5-8 Project Explorer with `DonateJPAEmployee`

#### Behind the scenes:

- ▶ The `persistence.xml` file defines the persistence unit, which by default is the project name.
- ▶ The `orm.xml` file can be used to define explicit mappings of JPA entities to database tables.

## 5.5.2 Project dependencies

In this section you verify that the JPA Project wizard added `DonateJPAEmployee` JPA project to the `DonateEAR` enterprise application project.

Note that the `DonateEAR` deployment descriptor, `application.xml`, still shows error the error that it must contain one or more modules in spite of the fact that `DonateJPAEmployee` has been added. This is because it was added as a dependent JAR file and not as a Java EE module. (web, EJB, client, connector).

Java EE does include the JPA specification, but JPA is not uniquely tied to any specific Java EE module type. Therefore, most developers typically include JPA entities in a general project archive that can be referenced from a formal Java EE module, and tooling platforms such as Eclipse and Rational Application Developer support this pattern.

1. In the Enterprise Explorer, select **DonateEAR**, right-click and select **Properties**.

2. At the Properties pop-up, on the right select **Java EE Module Dependencies**.
3. On the right at the resulting Java EE Module Dependencies pane (Figure 5-9 on page 162), you can observe that DonateJPAEmployee.jar has been added as a dependency.
  - a. Click **OK** to close the Properties pop-up.

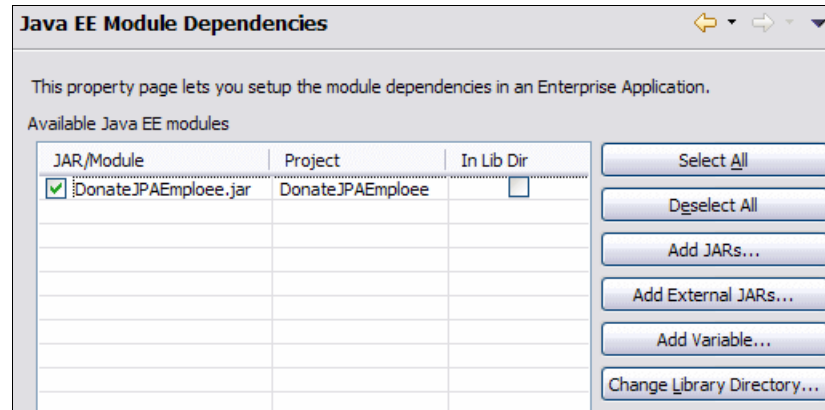



Figure 5-9 Module Dependencies pane

## 5.6 Generate the Employee entity from a table

In this section we first create the Employee entity and then verify the persistence.xml file.

### 5.6.1 Create the Employee entity

Here we create the Employee entity from the existing EMPLOYEE table in the Vacation database, using the *bottom-up scenario*:

1. Switch to the JPA perspective (it should be open already):
  - a. Click the **Select Perspective** icon [  ] located at the upper right, and select **Other** from the pull-down.
  - b. At the Open Perspective pop-up, select **JPA** click **OK**.
2. In the Data Source Explorer view (by default in the lower left of the JPA perspective), ensure that the **Vacation** database has an active connection (Figure 5-10 on page 163).

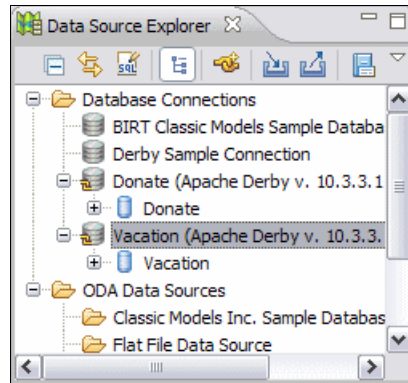


Figure 5-10 Data Source Explorer showing Vacation database

### Off course?

- ▶ If the Vacation database is not in the list of connections, repeat the steps in 4.4, “Create and configure the Vacation database” on page 90.
- ▶ If you cannot connect, then make sure you start the Derby network sever as described in 4.2, “Start the Derby Network Server” on page 87
- ▶ If the Vacation connection cannot be expanded, then select **Vacation**, right-click and select **Connect**.

These issues can occur if you closed the view or restarted the workspace or system after following the steps in Chapter 4, “Prepare the legacy application” on page 85.

3. In the Project Explorer (by default in the upper left of the JPA perspective), select **DonateJPAEmployee**, right-click and select **JPA Tools** → **Generate Entities**.
4. At the Generate Entities/Database Connection pop-up (Figure 5-11 on page 164), make sure that the **Vacation** connection and **VACATION** schema are selected, and click **Next**.

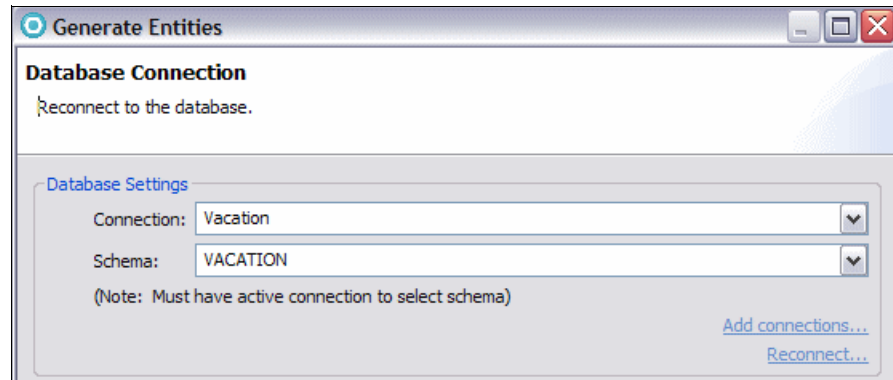


Figure 5-11 Generate Entities

**Off course?** Ensure that you are connected to the Vacation database. Otherwise the entities will not be generated. To connect to it open the Data Source Explorer view and double click on the vacation connection (or right click and select Connect).

5. At the Generate Entities/Generate Entities from Tables pop-up (Figure 5-12 on page 165)
  - Set the Package to `vacation.entities`.
  - a. Select **Synchronize Classes in persistence.xml**.
  - b. Select the **EMPLOYEE** table.
  - c. Click **Finish**.



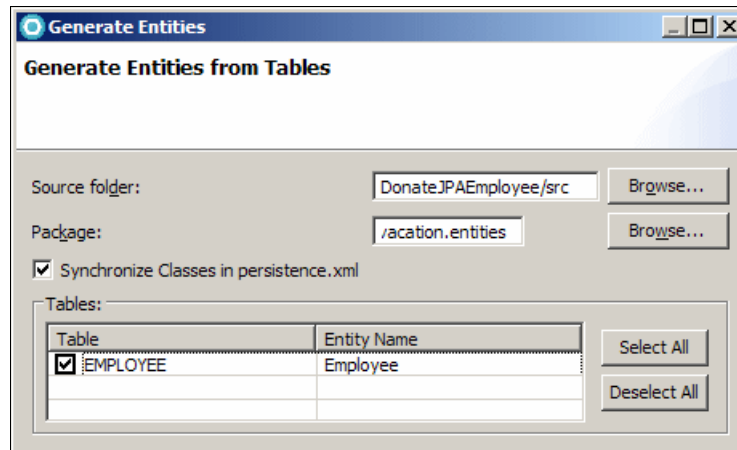


Figure 5-12 Generate Entities from Tables: Employee table

6. In the Enterprise Explorer, double-click **Employee.java** (in DonatJPAEmployee /src/vacation.entities) to open the file.
7. In the resulting Employee.java editor, note that the JPA tooling has generated the Java class fields and accessor methods automatically. These fields correspond to the columns of the EMPLOYEE table in the database. Example 5-5 shows an extract of the generated Employee class.
8. Close Employee.java when finished.

*Example 5-5 Employee entity (extract)*

---

**@Entity**

```
public class Employee implements Serializable {
    @Id
    @Column(name="EMPLOYEE_ID")
    private int employeeId;

    @Column(name="LAST_NAME")
    private String lastName;

    private BigDecimal salary;

    .....

    public int getEmployeeId() {
        return this.employeeId;
    }

    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }
}
```

```
}  
.....  
}
```

---

**Behind the scenes:** In addition to fields and accessor methods, the JPA tooling also adds JPA annotations, such as `@Entity`, `@Id`, and `@Column` automatically.

Because SQL database field names are case-insensitive, the JPA tooling assumes that words are separated by underscores. Thus, column names (`FIRST_NAME`) are mapped to Java fields (`firstName`) using the lowerCamelCase convention of Java. For these fields, a `@Column` annotation is added to map the field to the database column. The `salary` field does not require a `@Column` annotation because the column name is the same.

The `@Id` annotation defines the primary key of the entity

## 5.6.2 Update the persistence.xml file

Because we selected **Synchronize Classes in persistence.xml**, the `Employee` entity is added to the `persistence.xml` file.

The only information that is missing the mapping to the datasource that will be used in the deployed application. This specific JPA configuration can be challenging because the JPA entity can be deployed to multiple environments (standalone JUNIT or ExperienceJEE Test Server) and can be JTA (transactional) or non-JTA (non-transactional).

This book is only utilizing JTA data sources so the configuration is somewhat simplified, but the JPA project is still used in two environments, and that requires multiple `persistence.xml` configurations.

The approach used below configures the `DonateJPAEmployee` `persistence.xml` for deployment to the ExperienceJEE Test Server, referencing the `jdbc/Vacation` data source defined in 4.6.4, “Create the Vacation JDBC data source” on page 112. This configuration will first be used in Chapter 6, “Create the entity facade session beans” on page 201 when you deploy `DonateEAR` to the ExperienceJEE Test Server.

For the JUNIT test, a separate `persistence.xml` file will be created in the `DonateUnitTest` project and programmatically loaded to override the `DonateJPAEmployee` `persistence.xml`.

1. In the Project Explorer, open **persistence.xml** (in DonateJPAEmployee /src/META-INF).
2. In the resulting persistence.xml editor, select the Design tab (on the bottom left of the editor). Note that a persistence unit named DonateJPAEmployee has been created, with the Employee entity contained in it (Figure 5-13).

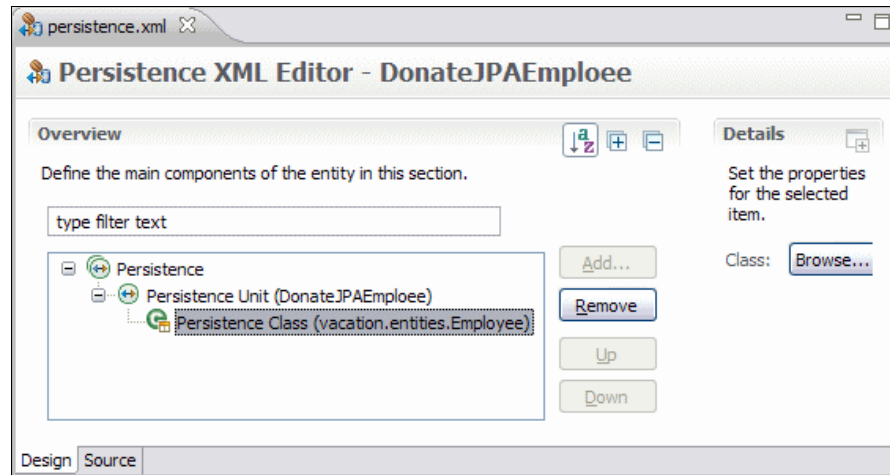


Figure 5-13 Persistence XML Editor: DonateJPAEmployee

3. Select the source tab and insert the jta-data-source statement before the class statement:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence ...>
  <persistence-unit name="DonateJPAEmployee">
    <jta-data-source>jdbc/Vacation</jta-data-source>
    <class>vacation.entities.Employee</class>
  </persistence-unit>
</persistence>
```

**Behind the scenes:** our application accesses the database only through JTA transactions. We could also define a `<non-jta-data-source>` for access by non-transactional programs.

4. Still at the source tab, insert the following statements after the class statement:

```
– Save and close persistence.xml

<class>vacation.entities.Employee</class>
<properties>
  <property name="openjpa.ConnectionUserName" value="VACATION" />
```

```
</properties>
</persistence-unit>
</persistence>
```

**Behind the scenes:** the connection user name is specified to force the Employee table to be created in the vacation schema. We could have also used the `openjpa.jdbc.Schema` property, but opted to use the connection user name since that is consistent with the Derby convention of using the connection user name as the default schema.

## 5.7 Create the Fund entity

In 5.6, “Generate the Employee entity from a table” on page 162, we used an existing database table to generate a JPA entity. In this section, we create a JPA entity from scratch. This class will be used to generate a database table in the Donate database using the *top-down scenario*.

From the viewpoint of the Experience Java EE scenario, this step represents the addition of new capability to the application.

### 5.7.1 Create the DonateJPAFund project

Data for donation funds will be stored in the Donate database. We create a JPA entity named Fund for this purpose.

We already have one JPA project for entities of the Vacation database, and we could create the Fund entity in the same DonateJPAEmployee project. This would result in two persistence units in the same project (in the same `persistence.xml` file).

The JPA runtime architecture does support having multiple persistence units in the same `persistence.xml`. However, one side effect is that the persistence manager will create tables for all the persistence unit in each and every data source. In our case, this would result in an extra `VACATION.FUND` table in the Vacation database, and an extra `DONATE.EMPLOYEE` table in the Donate database. These tables are created with the wrong schema and are in fact never used or accessed by the application.

In addition, the JPA tooling does not formally support multiple persistence units in the same `persistence.xml`:

- The tooling displays the following warning whenever validating a `persistence.xml` with multiple persistence units:

Multiple persistence units defined - tooling only supports 1 persistence unit per project

- JPA validation only works against one database connection and thus in theory would generate errors, because only one table would be found. In reality, due to the duplicate table problem described previously, validation would work because both tables exist in both databases.

Therefore, to avoid these issues, we create a second JPA project named `DonateJPAFund`. These steps are similar to those followed in 5.5.1, “Create the `DonateJPAEmployee` project” on page 157:

1. In the Project Explorer, select anywhere in the whitespace, right-click and select **New** → **Other**.
2. At the New pop-up, select **JPA** → **JPA Project** and click **Next**.
3. At the New JPA Project/JPA Project pop-up:
  - Set the project name to **DonateJPAFund**.
  - Set the configuration to **Default Configuration for WebSphere Application Server v7.0**.
  - a. Select **Add project to an EAR**.
    - Set Ear Project Name to **DonateEAR**.
  - b. Click **Next**.
4. At the New JPA Project/JPA Facet pop-up:
  - Set Platform to **RAD JPA Platform**.
  - Set Connection to **Donate**.
  - Select **Use implementation provided by server runtime**.
  - Select **Discover annotated classes automatically**.
  - Select **Create orm.xml**.
  - a. Click **Finish**.

## 5.7.2 Create the Fund class

Follow these steps to create the Fund class that will be the basis for the Fund entity.

1. In the Project Explorer, select **DonateJPAFund**, right-click and select **New** → **Class**.
2. At the New Java Class/Java Class pop-up:
  - Set Package to **donate.entities**

- Set Name to **Fund**.
  - For Interfaces, click **Add**.
    - At the Implemented Interfaces Selection pop-up, search and select **java.io.Serializable** and click **OK**.
  - a. Click **Finish** to create and open the Fund class in the Java Editor.
3. In the class body (after the public class definition and before the closing brace), declare the two java variables name and balance:

```
public class Fund implements Serializable {
    protected String name;
    protected int balance;
}
```

4. Select anywhere in the editor, right-click and select **Source** → **Generate Getters and Setters**.
- At the Generate Getters and Setters pop-up, click **Select All** and click **OK**

#### Behind the scenes:

The JPA specification does not explicitly require accessor methods for its entities, but they are generally recommended. Not only do they provide encapsulation of any business logic such as data validation, they also pave the way for loose coupling.

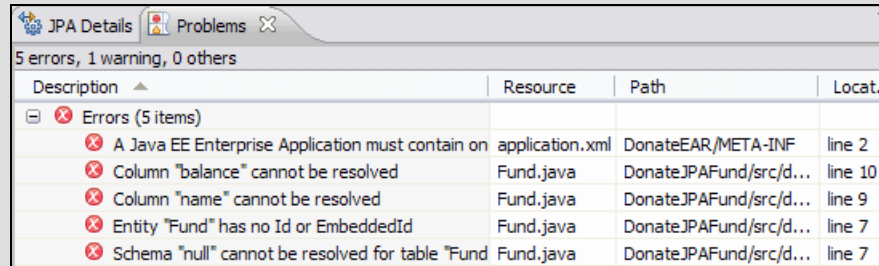
However, there are a few cases where you might not want to use getters and/or setters. For example, if you wanted to ensure that an entity was immutable (cannot be changed after it is created), you can elect to omit setter methods. If you omit accessor methods, JPA will try to access the fields directly. Thus, you must declare the fields public so they are visible to JPA.

5. Before the public class definition, add the **@Entity** annotation:

```
@Entity
public class Fund implements Serializable {
```

### Behind the scenes:

Note the several new errors for the Fund class, indicating various errors related to this JPA entity. These errors are present because the RAD JPA validator cannot locate the matching schemas and table for the Fund entity. This table and schema will be created when you run the entity for the first time in 5.8.4, “Run the JUnit test” on page 186 and the validation errors will be resolved at that point.



The screenshot shows the Eclipse IDE's Problems view. The title bar indicates 'JPA Details' and 'Problems'. Below the title bar, it says '5 errors, 1 warning, 0 others'. The table below lists the errors:

Description	Resource	Path	Location
Errors (5 items)			
A Java EE Enterprise Application must contain on	application.xml	DonateEAR/META-INF	line 2
Column "balance" cannot be resolved	Fund.java	DonateJPAPFund/src/d...	line 10
Column "name" cannot be resolved	Fund.java	DonateJPAPFund/src/d...	line 9
Entity "Fund" has no Id or EmbeddedId	Fund.java	DonateJPAPFund/src/d...	line 7
Schema "null" cannot be resolved for table "Fund"	Fund.java	DonateJPAPFund/src/d...	line 7

6. Select **Source** → **Organize Imports** (or press Ctrl+Shift+O) to resolve `javax.persistence.Entity`.
7. Leave the editor open because we will review the file in the next section after running the Configure JPA Entities wizard.

### 5.7.3 Configure the Fund entity

The configuration of a JPA entity can be done via manual edits in the various project source files or via the workbench graphical wizard. In this section you use the wizard to perform the configuration and the behind the scenes sections describe the resulting changes in the source files.

1. In the Project Explorer, select **Fund** (in `DonateJPAPFund/src/donate.entities`), right-click and select **JPA Tools** → **Configure JPA Entities**.
2. At the resulting Configure JPA Entities/Configure JPA Entities pop-up (Figure 5-14 on page 172) select **Fund** and click **Next**.

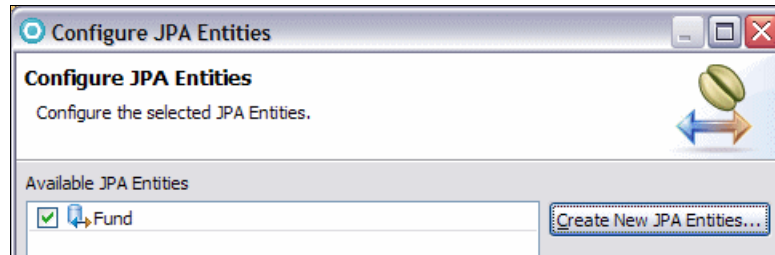


Figure 5-14 Configure JPA Entities: Fund

**Behind the scenes:** the fund entity is displayed because of the @Entity annotation that you added to the class in the previous section.

3. At the Configure JPA Entities/Tasks pop-up, on the left, select **Primary Key** and on the right enable the **name** attribute checkbox.

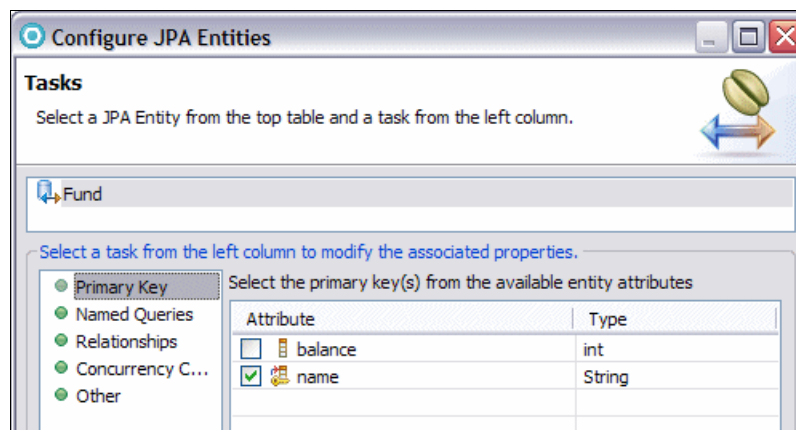


Figure 5-15 Configure JPA Entities: Tasks



### Behind the scenes:

The wizard adds the `@Id` annotation above the name variable type declaration in the Fund class to define it as the table primary key:

```
@Id
protected String name;
```

To keep things simple, we are using the fund name as the primary key. Most database administrators would likely frown on this in the real world, though, because the fund name could change. JPA supports a number of automatic identity generators. For more information about how this works, refer to “Entity automatic identity generation” on page 131

4. Still at the Configure JPA Entities/Tasks pop-up, on the left select **Named Queries** and on the right click **Add** (Figure 5-16 on page 173).
  - At the resulting Add Named Query pop-up:
    - In the upper section under the Result Attributes tab, select all the attributes (**name** and **balance**)
    - In the lower section under Query Statement, change the Named Query Name to **getAllFunds**.
  - a. Click **OK**.

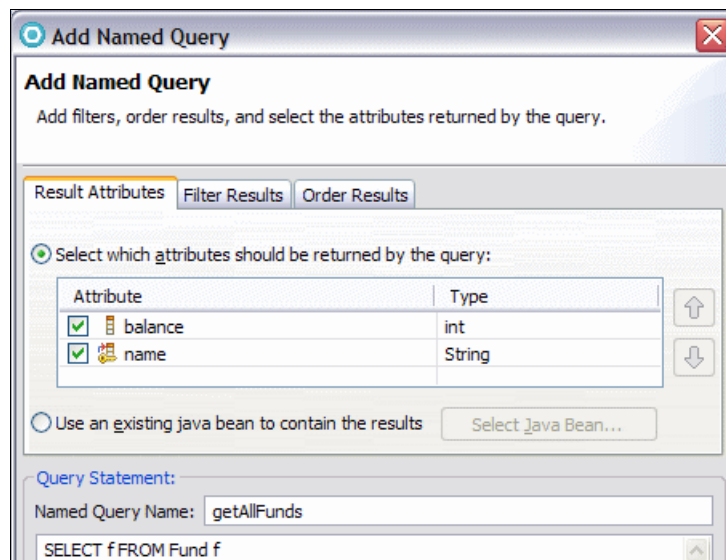


Figure 5-16 Configure JPA Entities/Tasks pop-up: Add named query

### Behind the scenes:

The wizard inserts an `@NamedQuery` annotation in the `Fund` class with the attributes entered in the dialog:

```
@NamedQuery(name="getAllFunds", query = "SELECT f FROM Fund f")
public class Fund implements Serializable {
```

5. Back at the Configure JPA Entities/Tasks pop-up (Figure 5-17 on page 174), ensure that the named query is present, and click **Finish**.

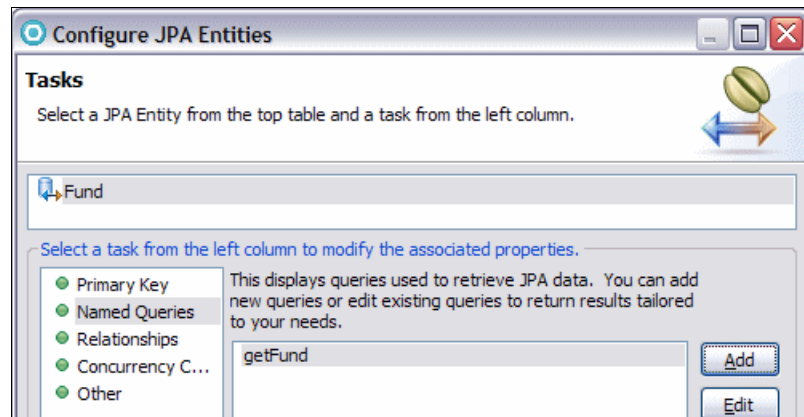


Figure 5-17 Configure JPA Entities/Tasks pop-up (verify named queries)

6. At the resulting Set up connections for deployment pop-up (Figure 5-18 on page 175):
  - a. Clear **Deploy JDBC Connection information to server**.
  - b. Enable **Set up persistence.xml**.
    - Enable **Use data source or resource reference**.
    - Set Name to **jdbc/Donate**.
    - Set Type to **JTA**.
  - c. Enable **Schema** and set to **DONATE**.
  - d. Click **OK**.

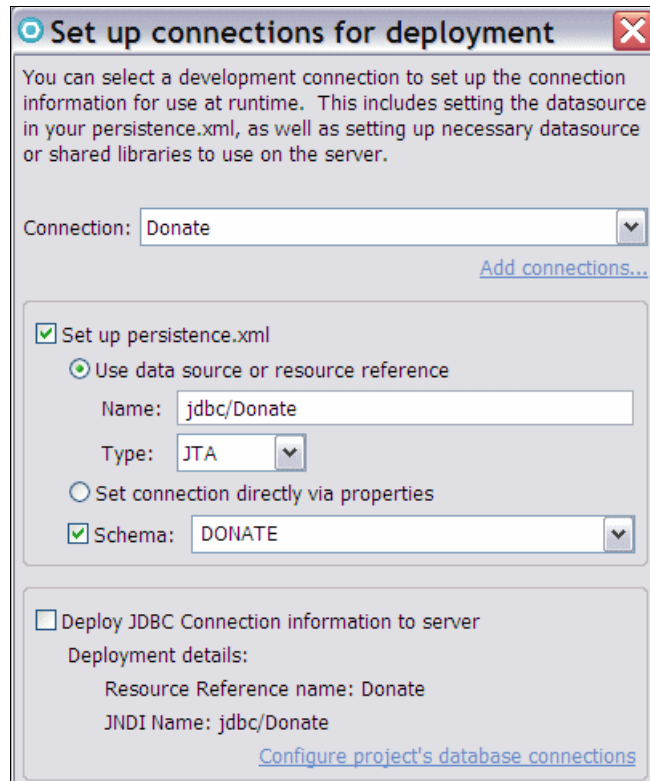


Figure 5-18 Set up connections for deployment pop-up

#### Off course:

If the Set up connections for deployment wizard did not automatically open, select the project in the Package Explorer, right-click and select **JPA Tools > Configure Project for JDBC deployment**.

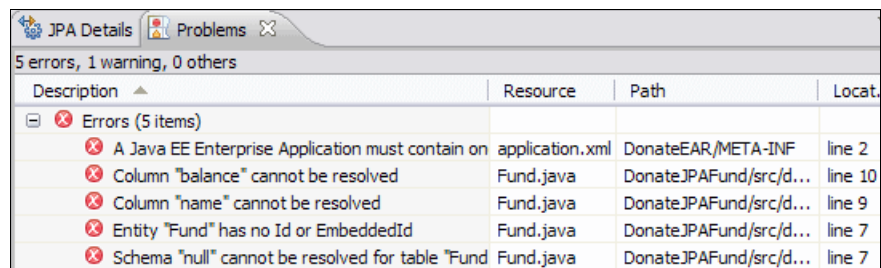
### Behind the scenes:

The Set up connections for deployment wizard adds to the persistence xml file the information to use the datasource configured in it. It configures the data-source name used by the persistence layer and the default schema name.

```
<persistence-unit name="DonateJPAFund">
  <jta-data-source>jdbc/Donate</jta-data-source>
  <class>donate.entities.Fund</class>
  <properties>
    <property name="openjpa.jdbc.Schema" value="DONATE"/>
  </properties>
</persistence-unit>
```

The Deploy JDBC Connection information creates a default (not distributed one phase commit) data source. We manually define the data source in the WebSphere Application Server in 4.6.5, “Create the Donate JDBC data source” on page 118. We deselect it otherwise it will override our definitions and causes the donate bean to fail when we execute it in 7.4, “Test the Donate session EJB with the UTC” on page 268

7. After the wizard completes (Figure 5-19), several errors will still exist relating to the mapping between the persistence unit and the database table. These errors will be resolved in subsequent sections.



JPA Details Problems			
5 errors, 1 warning, 0 others			
Description	Resource	Path	Location
Errors (5 items)			
A Java EE Enterprise Application must contain on	application.xml	DonateEAR/META-INF	line 2
Column "balance" cannot be resolved	Fund.java	DonateJPAFund/src/d...	line 10
Column "name" cannot be resolved	Fund.java	DonateJPAFund/src/d...	line 9
Entity "Fund" has no Id or EmbeddedId	Fund.java	DonateJPAFund/src/d...	line 7
Schema "null" cannot be resolved for table "Fund"	Fund.java	DonateJPAFund/src/d...	line 7

Figure 5-19 Errors displayed once wizard completes

### 5.7.4 Define the Fund entity database mapping

The JPA tooling provides several views that can be used to view and add annotated metadata to the classes, fields and definitions associated with JPA entities. You will use the following:

- The Fund java editor shows the entity source file

- The JPA Structure view provides a graphical representation of all the entities within the project
- The JPA Details view shows the properties of the selected entity.

We use these views to define the mapping between the JPA entity and the database.

1. In the upper right in the JPA Structure view (Figure 5-20), select donate.entities.Fund.

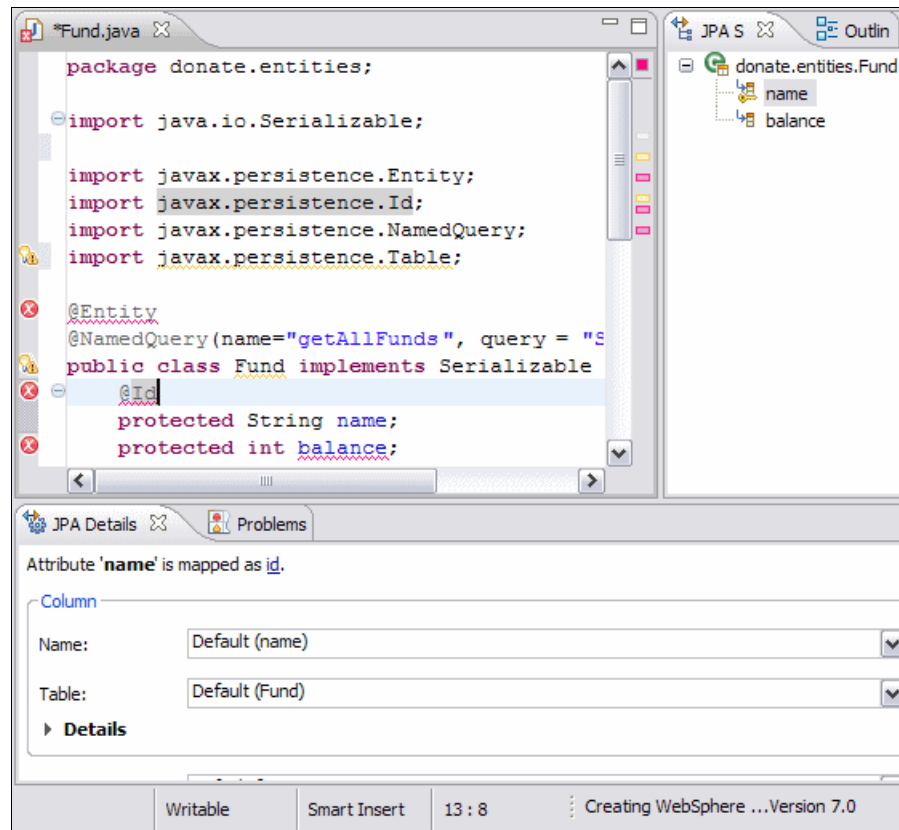


Figure 5-20 JPA Structure view

2. In the lower right in the JPA Details view, overwrite the JPA Details for the Table with **FUND** as table name, and **DONATE** as table schema (Figure 5-21 on page 178).

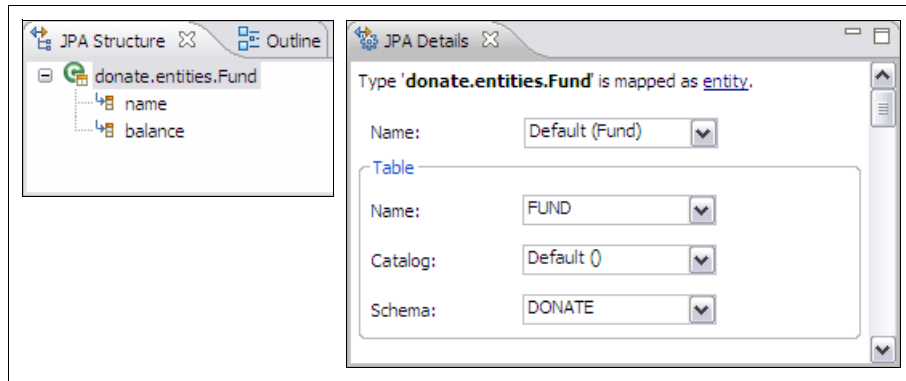


Figure 5-21 JAP Structure and JPA Details

#### Behind the scenes:

Note Fund.java is updated now contains a table annotation:

```
@Table(name="FUND", schema = "DONATE")
public class Fund implements Serializable {
```

3. Save the Fund editor (but do not close) to commit these changes. Ensure that the Editor name changes from \*Fund.java to Fund.java.

#### Behind the scenes

Several errors still exist because the Donate database does not have a table named Fund. This will be resolved in when the Fund Table is created when you run the JUNIT test in 5.8.4, “Run the JUnit test” on page 186.

### 5.7.5 Add the Fund entity to the persistence.xml

As mentioned in 5.1.7, “Persistence units” on page 141, the persistence.xml file provides the JPA implementation with details about how to connect to the underlying data sources. These connection details are grouped into *persistence units*.

We are using two distinct data sources in our application, Vacation and Donate. Thus, we require two distinct persistence units, one in each JPA project.

The bottom-up JPA tooling used in 5.6, “Generate the Employee entity from a table” on page 162 created a persistence unit automatically in DonateJPAAEmployee, containing a reference to the Employee class.

In this top-down section, we add the Fund class to the persistence.xml file in the DonateJPAFund project, and then configure the link to the data source.

1. In the Project Explorer, open **persistence.xml** (in DonateJPAFund /src/META-INF) (Example 5-6).
  - Switch to the source tab and note that it currently has no classes defined.
2. In the Project Explorer, select **persistence.xml** (in DonateJPAFund /src/META-INF), right-click and select **JPA Tools** → **Synchronize Classes**.
3. Back in the persistence.xml editor, note that a Funds class definition was added.
  - Save and close persistence.xml when done.

*Example 5-6 persistence.xml*

---

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

<persistence ... >
  <persistence-unit name="DonateJPAFund">
    <jta-data-source>jdbc/Donate</jta-data-source>
    <class>donate.entities.Fund</class>
    <properties>
      <property name="openjpa.jdbc.Schema" value="APP"/>
    </properties>
  </persistence-unit>
</persistence>
```

---

## 5.8 Test the Employee and Donate entities




In this section you test the JPA entities using JUnit.

JUnit is the most successful unit testing framework for Java. It allows test-driven development by making unit tests automate with simple reports. Additional information is available at:

- ▶ JUnit:  
<http://junit.sourceforge.net/>
- ▶ Rational Application Developer Information Center *Writing and running JUnit tests* topic:  
<http://publib.boulder.ibm.com/infocenter/rsasehlp/v7r5m0/topic/org.eclipse.jdt.doc.user/gettingStarted/qs-junit.htm>

## 5.8.1 Create the JUnit project

To test the entities, we create a JUnit project, which is basically a Java project:

1. Switch to the Java perspective:
  - a. Click the **Select Perspective** icon [  ] located at the upper right, and select **Other** from the pull-down.
  - b. At the Open Perspective pop-up, select **Java** and click **OK**.
2. In the workspace action bar, Select **File** → **New** → **Java Project**.
3. At the New Project/Create a Java project pop-up, set the project name to **DonateUnitTester**, and click **Next**.
4. At the New Java Project/Java Settings pop-up:
  - a. Select the **Projects** tab and click **Add**.
    - At the Required Project Selection pop-up, select **DonateJPAEmployee** and **DonateJPAFund**, and click **OK**.
  - b. Back at the New Java Project/Java Settings pop-up, select the **Libraries** tab and click **Add Library**:
    - At the Add Library/Add Library pop-up, select **JUnit** and click **Next**.
    - At the Add Library/JUnit Library pop-up, set the JUnit Library version to **JUnit 4** and click **Finish**,
  - c. Back at the New Java Project/Java Settings pop-up, select the **Libraries** tab and click **Add Library**:
    - At the Add Library/Add Library pop-up, select **Connectivity Driver Definition** and click **Next**.
    - At the Add Library/Connectivity Driver Definition pop-up, select **Derby 10.2 - Derby Client JDBC Driver Default** from the pull-down and click **Finish**.
  - d. Back at the New Java Project/Java Settings pop-up, select the **Libraries** tab and click **Add External Jars**:
    - At the JAR selection pop-up, add the following jar and click **Open**.
    -  **Windows**  
C:\IBM\SDP\runtimes\base\_v7\runtimes\com.ibm.ws.jpa.thinclient\_7.0.0.jar
    -  **Linux**  
/opt/IBM/SDP/runtimes/base\_v7/runtimes/com.ibm.ws.jpa.thinclient\_7.0.0.jar



**Behind the scenes:** This jar file above is the IBM Thin Client for Java Persistence API (JPA) it allows a Java SE client application to use the Java Persistence API (JPA) to store and retrieve persistent data through the application server.

- e. Back at the New Java Project/Java Settings pop-up, select the **Libraries** tab and click **Add External Jars:**

- At the JAR selection pop-up, add the following jar and click **Open**.

C:\IBM\SDP\runtimes\base\_v7\runtimes\com.ibm.ws.admin.client\_7.0.0.jar

**Behind the scenes:** This jar file above is the IBM Administration Think Client. It allows a Java SE client application to use the WebSphere administrative functions (such as the WebSphere initial context factory).

- f. Back at the New Java Project/Java Settings pop-up, select the **Libraries** tab and click **Add External Jars:**

- At the JAR selection pop-up, add the following jar and click **Open**.

C:\IBM\SDP\runtimes\base\_v7\runtimes\com.ibm.ws.ejb.thinclient\_7.0.0.jar

**Behind the scenes:** This jar file above is the IBM EJB thin client jar. It allows a Java SE client application to use the WebSphere EJB functions and contains additional classes needed to access WebSphere JAAS security.

The administrative and JPA libraries added in the two preceding steps actually contain a sufficient set of jars needed to access the session EJBs. However, this jar file is needed to allow the JUnit test to access the secured EJB in “10.5.4 Test the EJB declarative security using JUnit” on page 383.

- g. Back at the New Java Project/Java Settings pop-up, click **Finish**.

## 5.8.2 Create the persistence units for the Java SE profile

The Java Persistence API came out of the framework of the JSR 220 Expert Group (EJB 3.0). However, the Java Persistence API can be used outside the context of EJBs, including in a Web module running inside the application server

or even in a standalone Java application, such as JUnit, running outside the application server.

Because the JUnit tests are running outside of the application server, the `persistence.xml` file must define the JPA implementation to be used in the Java SE environment. We could modify the existing `persistence.xml` files, but it is much better to provide a `persistence.xml` file in the JUnit project itself. This `persistence.xml` file takes precedence over the files in the related JPA projects.

We follow these steps:

1. In the Enterprise Explorer, expand the **DonateUnitTester** project.
2. Select **src**, right-click and select **New** → **Folder**.
  - ▶ Type **META-INF** as name and click **Finish**.
3. Select **META-INF** (in `DonateUnitTester/src`), right-click and select **New** → **File**.
  - a. At the File pop-up, set File name to **persistence.xml** and click **Finish**. The `Persistence.xml` editor will automatically open.
4. Add the **Snippets** view to the Java perspective (**Window** → **Show View** → **Other** and then select **General** → **Snippets**).
5. Select anywhere in the white space of the `persistence.xml` editor.
6. In the Snippets view (lower right), expand the **ExperienceJEE** category, and double-click on **F05.1 DonateUnitTester Persistence** to populate the new `persistence.xml` file with the required configuration settings.
7. Save and close `persistence.xml`.

### Behind the scenes:

This persistence.xml file has two persistence units, one for each database. Instead of data source connections, properties are used to define OpenJPA connections to the two databases.

For example, to connect to the Vacation database:

```
<persistence-unit name="DonateJPAEmployee">
  <class>vacation.entities.Employee</class>
  <properties>
    <property name="openjpa.ConnectionURL"
      value="jdbc:derby://localhost:1527/Vacation" />
    <property name="openjpa.ConnectionDriverName"
      value="org.apache.derby.jdbc.ClientDriver" />
    <property name="openjpa.ConnectionUserName" value="VACATION" />
    <property name="openjpa.Log" value="SQL=TRACE" />
    <property name="openjpa.RuntimeUnenhancedClasses"
      value="supported"/>
  </properties>
</persistence-unit>
```

Note the openjpa.Log property:

```
<property name="openjpa.Log" value="SQL=TRACE" />
```

With the SQL=TRACE value, all the SQL statements that are executed are displayed in the Console. This enables you to monitor database access.

Note the openjpa.RuntimeUnenhancedClasses property:

```
<property name="openjpa.RuntimeUnenhancedClasses" value="supported" />
```

OpenJPA typically requires that the compiled persistence classes be enhanced to provide improved support. This can be done before execution by running the OpenJPA `PCEnhancer` command and certain environments (such as WebSphere Application Server) can dynamically enhance the classes when they are loaded.

However, this standalone java configuration does not support dynamic class enhancement and you cannot run the enhancement utility within the workbench (and you would have to export the project as a JAR file and then run the utility. Therefore, you instead use this property which instructs OpenJPA to support unenhanced classes.

Note that for the DonateJPAFund, a new database named Donate is created:

```
<property name="openjpa.ConnectionURL"
value="jdbc:derby://localhost:1527
/Donate;create=true" />
```

For the DonateJPAFund we use *runtime forward mapping*. We use one of these properties to tell OpenJPA to automatically generate any database tables that do not already exist. By enabling this feature, we are able to automatically generate the FUND table in the Donate database when its JPA entity is first accessed.

```
<property name="openjpa.jdbc.SynchronizeMappings"
value="buildSchema(ForeignKeys=true)" />
```

Because this persistence.xml file is in the DonateUnitTester project, the runtime uses it instead of the files in the JPA projects.

### 5.8.3 Create the JUnit test case

Next we create the test case to test the entities:

1. In the Enterprise Explorer, select **src** (in DonateUnitTester), right-click and select **New** → **Other** and then at the pop-up select **Java** → **JUnit** → **JUnit Test Case** and click **Next**.
2. At the New JUnit Test Case/JUnit Test Case pop-up (Figure 5-22 on page 184):
  - a. Select **New JUnit 4 test**.
  - b. Set the Package to **donate.test**.
  - c. Set the Name to **EntityTester**.
  - d. Click **Finish**.

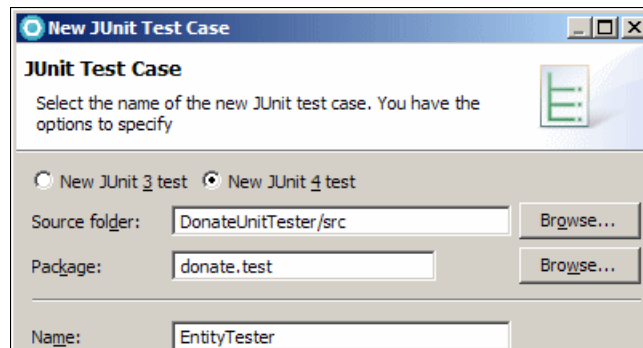


Figure 5-22 New JUnit Test Case pop-up

3. The `EntityTest` class opens in the editor:
  - a. Delete the existing lines.
  - a. In the Snippets view (lower right), expand the **ExperienceJEE** category, and double-click on **F05.2 EntityTester** to insert the complete JUnit test case.
4. Save and close `EntityTester.java`.

#### Behind the scenes:

- The `@BeforeClass` annotation for the `init` method is used to initialize the entity managers for the two persistence units:

```
public static void init() {
    emVacationUnit = Persistence
        .createEntityManagerFactory("DonateJPAEmployee")
        .createEntityManager();
    ....
}
```

- The `@Test` annotated `getEmployee` method retrieves an employee using the `find` method of `EntityManager`:

```
public void getEmployee() {
    Employee e = emVacationUnit.find(Employee.class, 2);
    .....
}
```

- The `@Test` annotated `createFund` method creates a fund using the `persist` method of `EntityManager`:

```
public void createFund() {
    int random = Math.abs(new Random().nextInt());
    Fund f = new Fund();
    f.setName("Test Fund #" + random);
    f.setBalance(20);
    emDonateUnit.getTransaction().begin();
    emDonateUnit.persist(f);
    emDonateUnit.getTransaction().commit();
}
```

- The `@Test` annotated `getAllFunds` method retrieves all the funds using the `getAllFunds` named query:

```
public void getAllFunds() {
    List<Fund> funds = emDonateUnit.createNamedQuery("getAllFunds")
        .getResultList();
    assertTrue("No funds found", funds.size() > 0);
    .....
}
```

## 5.8.4 Run the JUnit test

We are now ready to run the JUnit test:

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to:
    - **Windows** `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    - **Linux** `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`

and execute:

- **Windows** `startNetworkServer.bat`
- **Linux** `startNetworkServer.sh`

### Behind the scenes:

Recall that you started this in 4.2, “Start the Derby Network Server” on page 87 and it should still be running unless you stopped it restarted the system.

2. In the Enterprise Explorer, select **EntityTester.java** (in `DonateUnitTester/src/donate.test`), right-click and select **Run As → JUnit Test**.
3. The JUnit view opens automatically (Figure 5-23).
  - a. If the view opens in an inconvenient place, just drag the view to the bottom right.
  - b. Verify that all tests were successful, that is, the tests are marked by a green check mark.

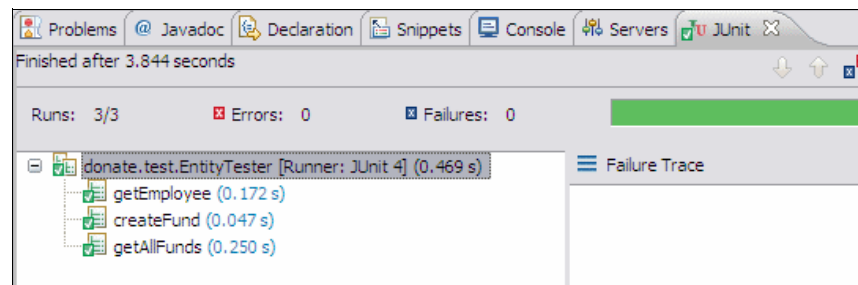


Figure 5-23 JUnit view

4. Add the **Console** view to the Java perspective (**Window** → **Show View** → **Other** and then select **General** → **Console**).
5. The Console view displays the detailed results of the test (Example 5-7 on page 187). Note that the standard output (normal logging) and standard error (SQL trace) are written independently and therefore the relative order of some of the statements are out of sequence.

*Example 5-7 JUnit test output with SQL statement trace*

---

```

Creating EntityManagers
=====
131 VacationUnit INFO [main] openjpa.Runtime - Starting OpenJPA
1.2.2-SNAPSHOT
291 VacationUnit INFO [main] openjpa.jdbc.JDBC - Using dictionary class
"org.apache.openjpa.jdbc.sql.DerbyDictionary".
2214 VacationUnit INFO [main] openjpa.Enhance - Creating subclass for
"[class vacation.entities.Employee]". This means that your application will
be less efficient ...[truncated]

10 DonateUnit INFO [main] openjpa.Runtime - Starting OpenJPA
1.2.2-SNAPSHOT
10 DonateUnit INFO [main] openjpa.jdbc.JDBC - Using dictionary class
"org.apache.openjpa.jdbc.sql.DerbyDictionary".
260 DonateUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
2119204432> executing stmtnt 288035115 CREATE TABLE DONATE.FUND (name
VARCHAR(255) NOT NULL, balance INTEGER, PRIMARY KEY (name))
290 DonateUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
2119204432> [30 ms] spent
300 DonateUnit INFO [main] openjpa.Enhance - Creating subclass for
"[class donate.entities.Fund]". This means that your application will be
less efficient ...[truncated]
DonateUnit EntityManager successfully created
3846 VacationUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
611525747> executing prepstmtnt 1645371922 SELECT t0.FIRST_NAME,
t0.LAST_NAME, t0.MIDDLE_NAME, t0.salary, t0.VACATION_HOURS FROM Employee t0
WHERE t0.EMPLOYEE_ID = ? [params=(int) 2]
3846 VacationUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
611525747> [0 ms] spent
771 DonateUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
1193559844> executing prepstmtnt 1179338315 INSERT INTO DONATE.FUND (name,
balance) VALUES (?, ?) [params=(String) Test Fund #291857670, (int) 20]
771 DonateUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
1193559844> [0 ms] spent
991 DonateUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
670509047> executing prepstmtnt 866988973 SELECT t0.name, t0.balance FROM
DONATE.FUND t0

```

```
1001 DonateUnit TRACE [main] openjpa.jdbc.SQL - <t 109577864, conn
670509047> [10 ms] spent

Retrieving info for employee 2
=====
Employee info successfully retrieved!
Employee:
org.apache.openjpa.enhance.vacation$entities$Employee$pcsubclass@39e239e2
Employee Id: 2
First Name: Neil
Last Name:
Last Name: Armstrong
Salary: 100000
Vacation Hours: 50

Persisting new Fund
=====
Fund successfully persisted!

Retrieving Funds from database
=====

Name      Balance
Test Fund #29185767020
```

---



### Behind the scenes:

Recall that you enabled trace in the persistence.xml file:

```
<property name="openjpa.Log" value="SQL=TRACE" />
```

As a result, the trace output shows the SQL statements that are executed and the elapsed time of execution.

- The FUND table is created in the Donate database. You will verify that the table was created in the next step.

```
CREATE TABLE DONATE.FUND (name VARCHAR(255) NOT NULL, balance INTEGER,  
PRIMARY KEY (name))
```

- The employee is retrieved and displayed.

```
SELECT t0.FIRST_NAME, t0.LAST_NAME, t0.MIDDLE_NAME, t0.salary,  
t0.VACATION_HOURS FROM Employee t0 WHERE t0.EMPLOYEE_ID = ?  
[params=(int) 2]
```

- A new fund is created with 20 hours.

```
INSERT INTO DONATE.FUND (name, balance) VALUES (?, ?) [params=(String)  
Test Fund #291857670, (int) 20]
```

- All the funds re displayed.

```
SELECT t0.name, t0.balance FROM DONATE.FUND t0
```

One fund is displayed the first time you run the JUnit test. Subsequent iterations will show more funds (because a new fund entry is created each time your run the test).

6. Verify the database activities using the workbench Data Source Explorer (Figure 5-24).
  - a. Switch to the Data Source Explorer view in the lower left of the current JPA perspective.
  - b. Refresh the Donate connection by selecting **Databases** → **Donate (Apache Derby xxx)** → **Donate**, right-click and select **Refresh**.
  - c. Verify that the FUND table was created and that a record was created.
  - d. Expand to **Databases** → **Donate (Apache Derby xxx)** → **Donate** → **Schemas** → **DONATE** → **Tables** → **FUND**. This will verify that the FUND database and table was created.
  - e. Select the FUND table, right-click and select **Data** → **Sample Contents**.
  - f. Switch to the SQL Results view (you might have to resize it), and then select the **Result1** tab on the left. An entry is displayed for every time the test was run, thus verifying the JUnit test.

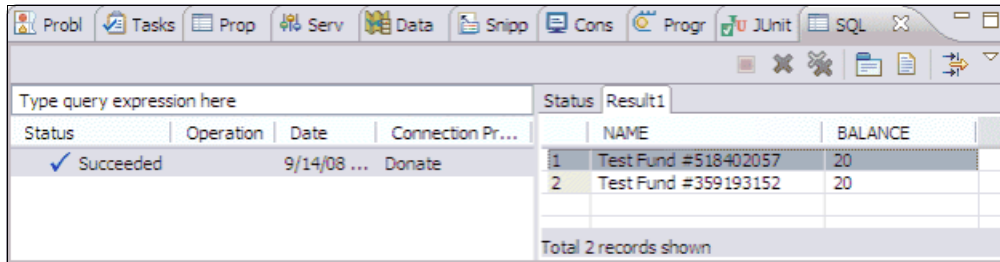


Figure 5-24 Data Source Explorer: verify the database activities

7. Since the Fund table now exists, the various JPA errors associated with the DonateJPAFund project can be removed by re-running validation (Figure 5-25).
  - a. Switch to the Problems view and verify that the three errors are still displayed for DonateJPAFund.

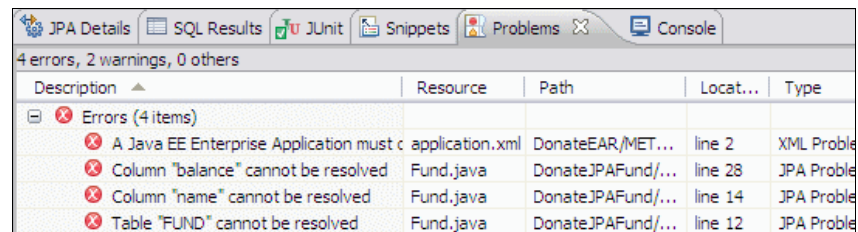


Figure 5-25 DonateJPAFund project errors

- b. In the Project Explorer, highlight **DonateJPAFund**, right-click and select **Validate**.
  - c. You should receive the following-up indicating that a successful validation (Figure 5-26). Click OK to close.

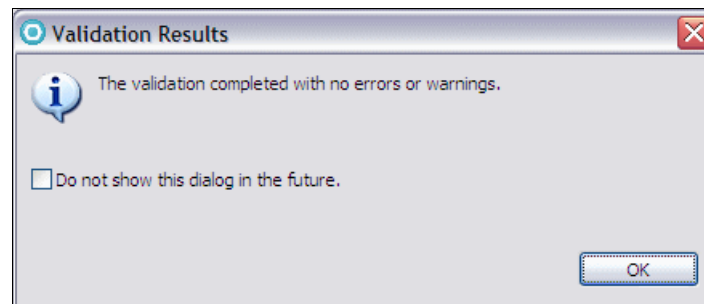


Figure 5-26 Validation Results

- d. Switch back to the Problems view and verify that the three errors related to `DonateJPAFund` are no longer displayed. Note that you still have the error related to `DonateEAR` (and that error will be resolved in Chapter 6, “Create the entity facade session beans” on page 201).

### Off course?

If the errors still exist, then the `Donate` database connection has not been refreshed in the workspace. Ensure that you completed the activities described in steps 6 on page 189.

### Alternative:

This section used the `openjpa.jdbc.SynchronizeMappings` directive in the `DonateUnitTester` to create the table the first time the database was active. What if you wanted to create the table ahead of time?

This can be done using standard Rational Application Developer capabilities:


- ▶ In the Java perspective Package explorer, select **DonateJPAFund**, right-click and select **JPA Tools** → **Generate DDL**.
- ▶ At the General DDL pop-up:
  - Ensure that **Use the defined database connection** is selected.
  - Set the DDL file name to **fund.ddl**.
  - Set the database schema to **donate**.
  - Click **Finish**.
  - At the resulting Information pop-up (indicating that `fund.ddl` was generated successfully) click **OK**.
- ▶ Back in then the Java perspective Package explorer, open the resulting **fund.ddl** (in `DonateJPAFund/src/META-INF`).
  - Select anywhere in the editor, right-click and select **Run SQ**.
  - At the resulting Select Connection Profile pop-up, select the **Donate** connection and click **Finish**.

## 5.9 Errors and warnings

By now the Problems view shows one error for the DonateEAR enterprise application, because we have no modules defined yet. In addition, you see a number of warnings, such as:

- ▶ XML warning: No grammar constraints (DTD or XML schema) are detected for the document (`application.xml`).
- ▶ Java warning: The serializable class `Fund` does not declare a static final `serialVersionUID` field of type `long` (`Fund.java`).

We can tailor the Problems view in a number of ways to reduce the number of errors and warnings.

1. By default, errors and warnings of all projects are shown. You can limit the Problems view to only show problems of one project (Figure 5-27 on page 193):
  - a. Click the **Show Menu** icon  (top right of the view), and select **Configure Contents**.

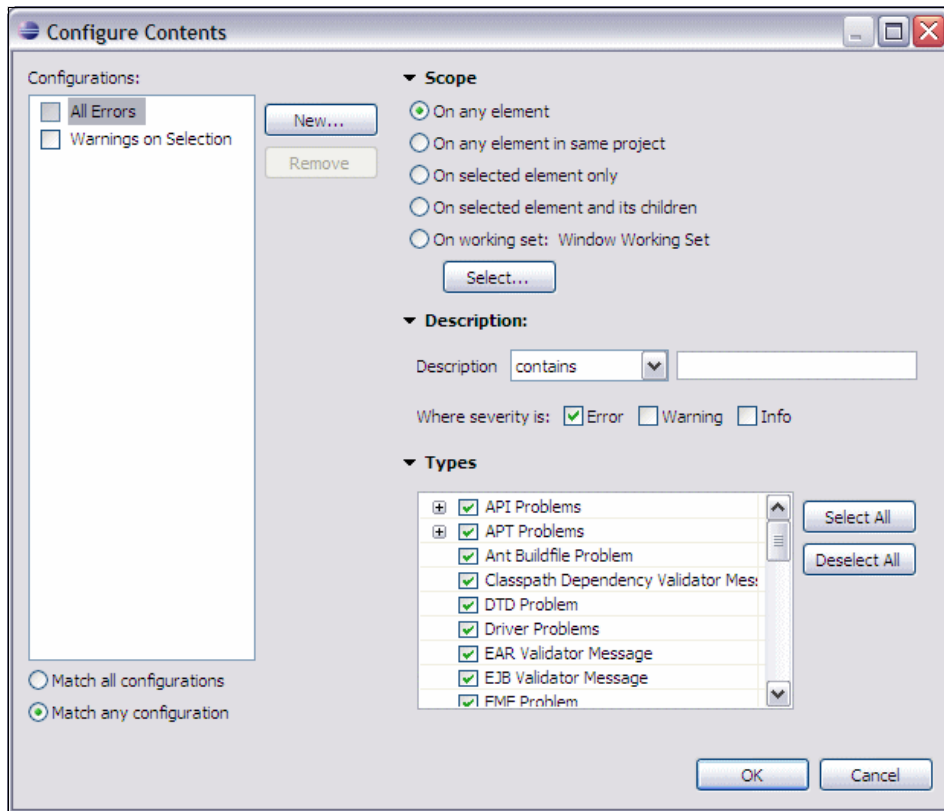


Figure 5-27 Limiting problems view

- b. Select **On any element in same project** to limit the view to a selected project.
  - c. Select **All Errors** (on the left) to remove all the warnings.
  - d. Select **Cancel** to close this pop-up without saving the changes. There are many other ways to be more selective about the problems shown in the Problems view. For now we will leave the default of showing all errors and warnings.
2. Resolve the warning No grammar constraints (DTD or XML schema) detected for the document ... by disabling this type of warning globally (for all schema files that lack grammar constraints):
  - a. In the workbench action bar, select **Window** → **Preferences**.
  - b. At the Preferences pop-up:
    - i. On the left, select **XML** → **XML Files**.

- ii. On the right (in the resulting XML Files pane), change the Validating files/Indicate when no grammar is specified from **Warning** to **Ignore**.
- iii. Click **OK**.

#### Behind the scenes:

XML validation errors can only be disabled globally. There is no way to turn off this type of warning for a single instance or project.

3. Resolve the warning: The serializable class Fund does not declare a static final serialVersionUID field of type long (Fund.java) by updating the code so that the warning is no longer issued.(Figure 5-28)
  - a. Double-click the warning, and the Fund class opens.
  - b. Right-click on the warning icon at the left border, and select **Quick Fix**.
  - c. Double-click **Add default serial version ID** (the first suggestion) and a serial version ID is created:
 

```
private static final long serialVersionUID = 1L;
```
  - d. Save and close Fund.java. The warning should disappear from the Problems view, and the warning disappears.

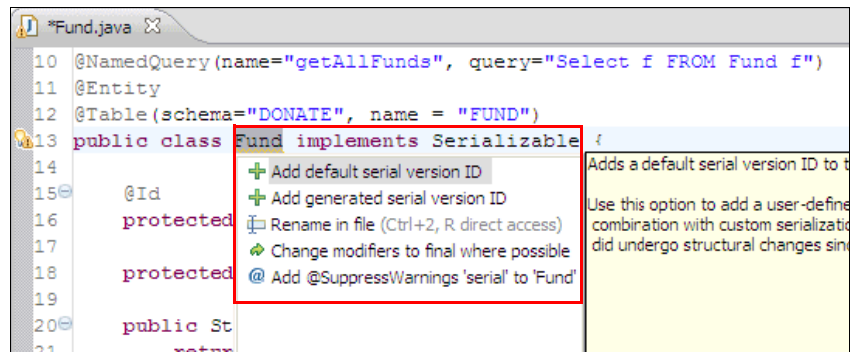


Figure 5-28 Resolve warning message

### Alternatives:

You can also access the quick fix menu by:

- ▶ Left-clicking on the warning icon on the left border.
- ▶ Selecting the line with the error and clicking CTRL+1.

You could also have removed this warning by:

- ▶ Accepting the quick fix recommendation to suppress this warning for this instance using a `@SuppressWarnings` annotation:

```
@SuppressWarnings("serial")
@NamedQuery(name="getAllFunds", query="Select f FROM Fund f")
@Entity
@Table(schema="DONATE", name = "FUND")
public class Fund implements Serializable {
```

- ▶ Globally disabling this warning for all instances through the Java compiler settings:
  - Open the Preferences pop-up (**Windows** → **Preferences**) and expand to **Java** → **Compiler** → **Errors/Warnings**.
  - In the Errors/Warnings pane (on the right), expand **Potential programming problems** and change Serializable class without serialVersionUID from **Warning** to **Ignore**.

Over time, you probably will use all of these approaches to fix or control the display of errors and warnings:

- ▶ Updating the code to resolve the error or warning (as we did here) generally is considered the best practice.
- ▶ You might not be able to easily update the code to resolve the warning, or perhaps in that specific situation the code is correct. In that case, you could choose to suppress that single instance.
- ▶ You might find that the particular error or warning exists in many of your files (or in the auto-generated files) and that the simplest approach is to ignore all of errors or warnings of that type. In that case, you can update the workbench to globally ignore the error or warning.

4. Optionally resolve the warning: Type safety: Unchecked cast from List to List<Fund> by using the Quick fix capability used in step 3 on page 194 to add the `@SuppressWarnings("unchecked")` annotation before the `getAllFunds` method.

## 5.10 Explore!

In this section we provide a few advanced topics for persistence.

### 5.10.1 Advanced topics for persistent entities

The object-oriented paradigm is largely used in application development in the industry. It models the world into application objects defining its behavior and state. Application objects need to be persisted in case of a system crash or a power loss. Databases are used as the persistence layer of most of the applications, however, even if there are some object oriented databases systems on the market, relational databases are more pervasive. It means that the object oriented systems must map their objects to relational model. This mapping can be done ad hoc for each application, or by using an object-relational mapping (O/RM) framework.

Using an O/RM framework or API reduces the coupling between the application and the relational persistence layer by making it transparent. It also allows you to improve performance by having smart fetching, better inheritance mapping strategies, automatic dirty checking, and other mapping techniques.

The complex approach that was used to map the objects in EJB 2.x prompted the creation of a simpler persistence framework and APIs, such as Hibernate, iBatis, and Oracle TopLink.

Java EE 5 specifications defined EJB 3.0 specification (JSR 220), and its major enhancement is the addition of the Java Persistence API (JPA). JPA defines how Java objects that have to be persisted (persistent entities) are mapped to relational data to keep their state.

JPA standardizes the object-relational mapping. It makes large use of annotations, and still allows deployment descriptors that override the annotations, making development easier, and allows deployers to customize the application without changing the code.

For further O/RM, JPA, and EJB 3.0 references, refer to the following Web addresses:

- ▶ Design enterprise applications with the EJB 3.0 Java Persistence API:  
<http://www.ibm.com/developerworks/java/library/j-ejb3jpa.html>
- ▶ Get to know Java EE 5:  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0707\\_barci/0707\\_barci.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0707_barci/0707_barci.html)



## 5.10.2 Experience advanced entities with JPA

In this chapter, we created object mappings for the simple entities (Employee and Fund). In this section we describe a more advanced object model and we show how to map it to a relational database using JPA.

We extended the object model defined in the core section to better illustrate the JPA mappings. The extensions were done in the DONATE database only, because based on the business model defined in the core chapter, the Vacation database is a legacy database and cannot be modified.

### Original mapping

The original mapping has the following entities:

Employee	Models the employee object; the employee can donate vacations to a vacation fund.
Fund	Models the fund to receive any donation of vacation hours from an employee.

The two entities are completely independent.

### Potential new mapping

The following entities are added:

FundAction	This abstract class models any action done on a Fund.
Donation	This class is a FundAction that represents any donation done by an employee to a fund.
Withdrawal	This class is a FundAction that represents a withdrawal made by an employee from a fund.
Approval	This is class that represents the information about the approval of a withdrawal.

The relationship between these classes are represented in a class diagram (Figure 5-29).

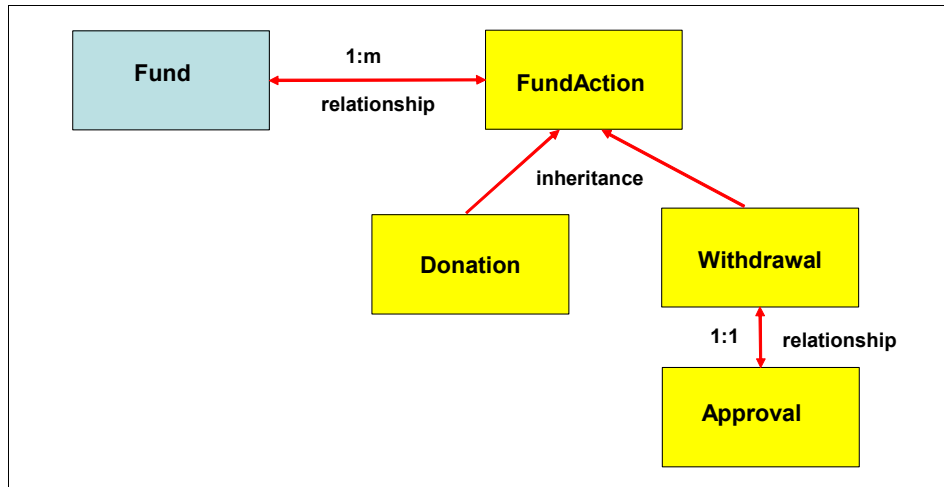


Figure 5-29 Advanced entity model with relationships and inheritance

## Entity relationships

JPA support relationships between entities. It supports one-to-one, one-to-many, many-to-one, and many-to-many relationships. Entity objects define other entities as collection members.

In our example, we have a one-to-many relationship between Fund and FundAction, that is, a fund can have many actions. We also have a many-to-one relationship between FundAction and Fund, that is, an action is for one fund.

These relationship can be expressed using collections and annotations in the Fund and FundAction classes (Example 5-8).

### Example 5-8 Relationship annotations

```

@Entity
public class Fund implements Serializable {

    @Id
    protected String name;

    protected int balance;

    @OneToMany(mappedBy="fund")
    private Set<FundAction> fundActionCollection;
    .....
  
```

```

=====
@Entity
public class FundAction implements Serializable {

    @Id
    private String id;

    int employeeId;

    private int hours;

    @ManyToOne
    private Fund fund;

    .....

```

---

## Inheritance

A fund action (FundAction class) can either be a donation or a withdrawal. This can be expressed using inheritance, that is, the Donation and the Withdrawal classes are subclasses of FundAction.

As discussed in 5.1.6, “Entity inheritance” on page 139, inheritance can be implemented using a single table, or multiple tables. In a single table approach, a discriminator column is used to distinguish between the different entity types.

Example 5-9 shows how a FUNDATION table with a column named action can be used to map the three entities.

### *Example 5-9 Inheritance annotations*

---

```

@Entity
@Table (name="FUNDATION")
@Inheritance
@DiscriminatorColumn(name="action",\
                    discriminatorType=DiscriminatorType.STRING, length=8)
@DiscriminatorValue("FundAction")
public class FundAction implements Serializable {

    @Id
    private String id;

    private int hours;

    @ManyToOne
    private Fund fund;

    .....

```

```
=====

@Entity
@Table (name="FUNDATION")
@Inheritance
@DiscriminatorValue("Donation")
public class Donation extends FundAction {

    .....

=====
```

```

@Entity
@Table (name="FUNDATION")
@Inheritance
@DiscriminatorValue("Withdraw")
public class Withdrawal extends FundAction {

    String reason;

    @OneToOne
    public Approval getApproval() {
        return approval
    };

    .....

```

---

**Note:** We did not implement this scenario.

For examples using relationship and inheritance, refer to the IBM Redbooks publication, *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611, or *Rational Application Developer V7.5 Programming Guide*, SG24-7672.



## Create the entity facade session beans

In this chapter, we create the EJB 3.0 session beans, `EmployeeFacade` and `FundFacade`, that are facades to the two JPA entities, `Employee` and `Fund` for our sample application.

## 6.1 Learn!

Figure 6-1 shows the session bean `EmployeeFacade` and `FundFacade` that are facades to the two JPA entities, `Employee` and `Fund` for our sample application.

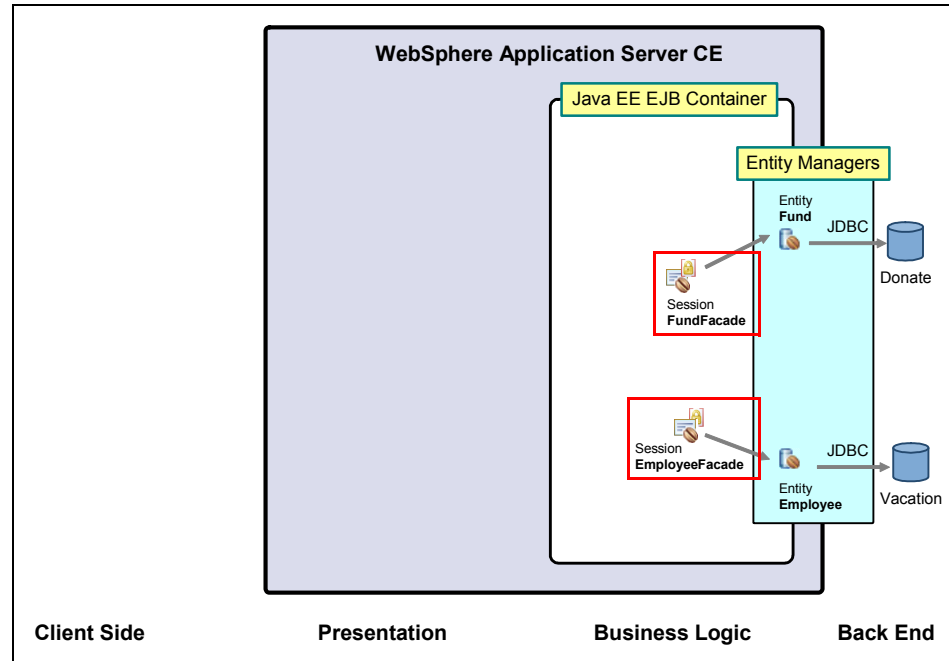


Figure 6-1 Donate application: session facades for JPA entities

JPA entity objects, such as those you created in the preceding chapter, encapsulate data into a record-oriented format. The next challenge is, how do you link these objects together to form a business function that acts against one or more of these entities?

Typically, the application clients should not access JPA entities directly. The answer is a session EJB that provides an execution environment for the business logic. The EJB container provides the core services such as transactional support and local/remote invocation, allowing the user-written code to focus on the business-level functions.

The end application accesses the session EJB using local and remote invocation mechanisms. The user-written session EJB methods use the JPA entities to access the underlying databases.

The session EJB method is run as a single transaction, and when the method completes all actions against the various JPA entities (such as creating or deleting, or changing values) all are committed or rolled back. The user does not have to insert any code to handle this transactional activity.

Session EJBs, through the user-written code, access the associated JPA entities through an entity manager.

Session EJBs can also be used in situations with no entities to execute other transactional (JMS messaging, JCA adapters, user-written code with transactional contexts) and non-transactional (JavaBeans or Web services) artifacts. You still benefit from the following features:

- ▶ Local and remote invocation, along with the associated failover and workload management.
- ▶ Common access and security control across all technologies (Web front end, Java EE application client, Web services, and messaging).

Finally, there are two types of session EJBs:

- ▶ **Stateful**, where the EJB maintains state across a series of invocations, thus making the results of one call dependent on a previous call.  
For example, the first call can initialize a variable that is used in subsequent calls.
- ▶ **Stateless**, where the EJB does not maintain state, thus making each call completely independent from any other call.

Typical EJB best practices almost always recommend against stateful session EJBs, because the complexity they add to workload management and failover far outweigh their benefits and because most practical applications do not need this support.

If you do have an application that has to maintain state or conversational context across session EJB invocations, an alternate design pattern is to store the state information in an entity and to have the stateless session EJB return an index or key to the requester on the first call. The requester provides this index/key on subsequent calls, and the stateless session EJB can use this to retrieve the state from the entity.

Additional Learn resources are available at the following Web sites:

- ▶ Java EE Enterprise JavaBeans technology:  
<http://java.sun.com/products/ejb/>
- ▶ Mastering Enterprise JavaBeans 3.0:  
<http://www.theserverside.com/tt/books/wiley/masteringEJB3/index.tss>

### 6.1.1 EJB 3.0 specification

In this section we learn about the EJB 3.0 specifications that describes session EJBs and message-driven EJBs. This section is an extract from the IBM Redbooks publication *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611, Chapter 1, Introduction to EJB 3.0.

### 6.1.2 EJB 3.0 simplified model

There are many publications that discuss the complexities and differences between the old EJB programming model and the new. For this reason, this book focuses on diving right into the new programming model. To overcome the limitations of the EJB 2.x, the new specification introduces a really new simplified model, whose main features are as follows:

- ▶ Entity EJBs are now plain old Java objects (POJO) that expose regular business interfaces (POJI), and there is no requirement for home interfaces.
- ▶ Removal of the requirement for specific interfaces and deployment descriptors (deployment descriptor information can be replaced by annotations).
- ▶ A complete new persistence model (based on the JPA standard), that supersedes EJB 2.x entity beans (see Chapter 5, “Create the JPA entities” on page 123).
- ▶ Interceptor facility to invoke user methods at the invocation of business methods or at life cycle events.
- ▶ Default values are used whenever possible (“configuration by exception” approach).
- ▶ Reduction of the requirements for usage of checked exception.

Figure 6-2 on page 205 shows how the model of J2EE 1.4 has been completely reworked with the introduction of the EJB 3.0 specification.



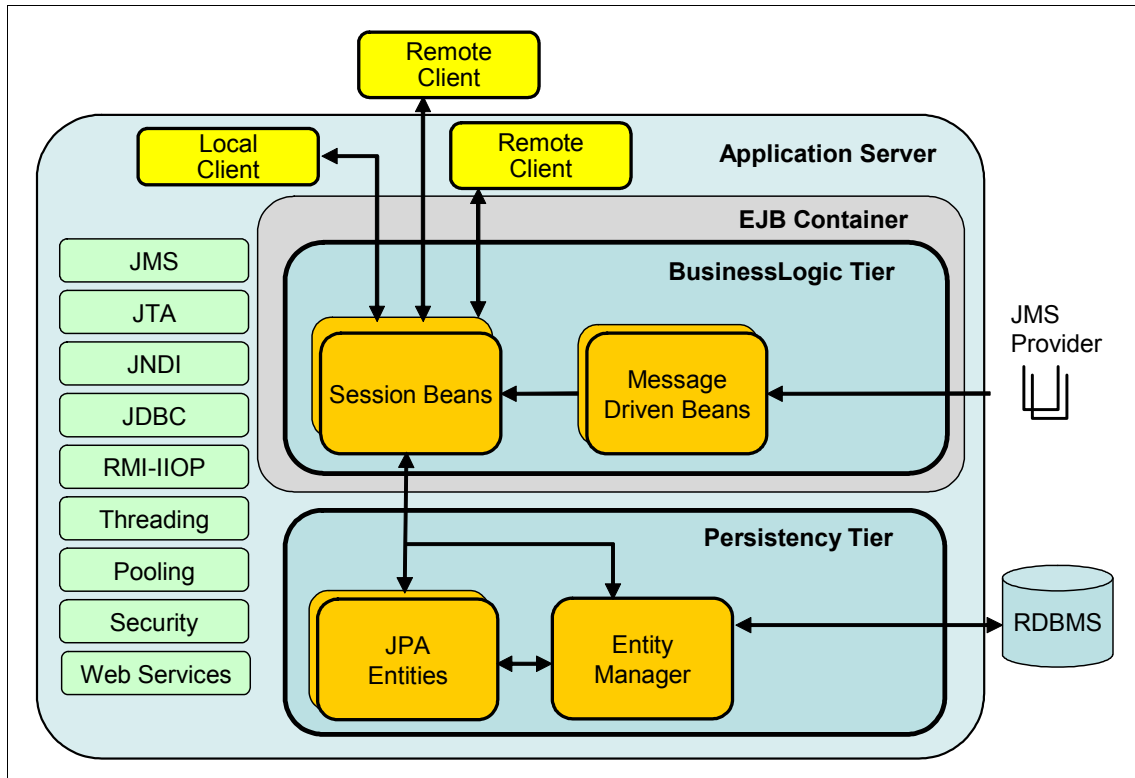


Figure 6-2 EJB 3.0 architecture

## Metadata annotations

EJB 3.0 uses *metadata annotations*, as part of Java SE 5.0.

Annotations are similar to XDoclet, a powerful open-source framework extensible, metadata-driven, attribute-oriented framework that is used to generate Java code or XML deployment descriptors. However unlike XDoclet, that requires pre-compilation, annotations are syntax checked by the Java compiler at compile-time, and sometimes compiled into classes.

By specifying special annotations, developers can create POJO classes that are EJB components.

In the sections that follow we illustrate the most popular use of annotations and EJB 3.0 together.

### 6.1.3 EJB types and their definition

In EJB 3.0 there are two types of EJBs:

- ▶ Session beans (stateless and stateful)
- ▶ Message-driven beans (MDB)

**Note:** EJB2.x entity beans have been replaced by JPA entities, which are discussed in Chapter 5, “Create the JPA entities” on page 123.

Annotations can be used to declare session beans and message-driven beans (MDB).

#### Stateless session EJBs

Session stateless EJBs have always been used to model a task being performed for a client code that invokes it. They implement business logic or rules of a system, and provide coordination of those activities between beans, such as a banking service, that allows for a transfer between accounts.

A stateless session bean is generally used for business logic that spans a single request and therefore cannot retain client-specific state among calls.

Because a stateless EJB does not maintain a conversational state, all the data exchanged between the client and the EJB must be passed either as input parameters, or as return value, declared on the EJB business method interface.

To better appreciate the simplification effort done by the EJB 3.0 specification, consider this comparison of the steps involved in the definition of an EJB according to 2.x and 3.0.

#### Steps to define a stateless session bean in EJB 2.x

To define a stateless session EJB for EJB 2.x, you have to define the following components:

- ▶ **EJB component interface:** Used by an EJB client to gain access to the capabilities of the bean. This is where the business methods are defined. The component interface is called the **EJB object**. There are two types of component interfaces (Figure 6-3 on page 207):
  - Remote component (EJBObject): Used by a remote client to access the EJB through the RMI-IIOP protocol.
  - Local component (EJBLocalObject): Used by a local client (that runs inside the same JVM) to access the EJB.

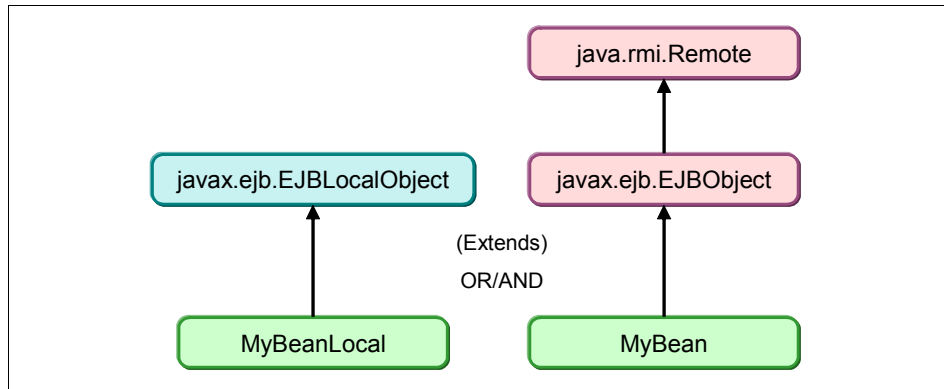


Figure 6-3 EJB 2.x component interfaces

- **EJB home interface:** Used by an EJB client to gain access to the bean. Contains the bean life cycle methods of create, find, or remove. The home interface is called the **EJB home**. The EJBHome object is an object which implements the home interface, and as in EJBObject, is generated from the container tools during deployment, and includes container specific code. At startup time, the EJB container instantiates the EJBHome objects of the deployed enterprise beans and registers the home in the naming service. An EJB client accesses the EJBHome objects using the Java Naming and Directory Interface (JNDI). There are two types of home interfaces (Figure 6-4):
  - Remote home interface (EJBHome): Used by a remote client to access the EJB through the RMI-IIOP protocol.
  - Local home interface (EJBLocalHome): Used by a local client (that runs inside the same JVM) to access the EJB.

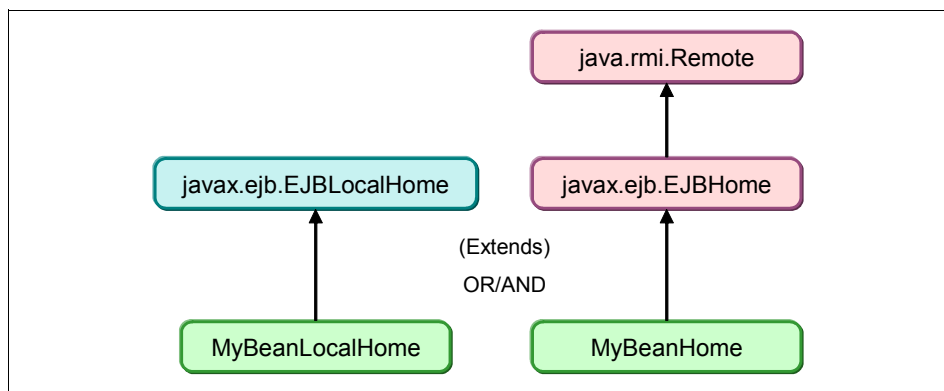


Figure 6-4 EJB 2.x home interfaces

- **EJB bean class:** Contains all of the actual bean business logic. Is the class that provides the business logic implementation. Methods in this bean class associate to methods in the component and home interfaces.

## Steps to define a stateless session bean in EJB 3.0

To declare a session stateless bean, according to EJB 3.0 specification, you can simply define a POJO:

```
@Stateless
public class MyFirstSessionBean implements MyBusinessInterface {

    // business methods according to MyBusinessInterface
    .....
}
```

Consider the new, revolutionary aspects of this approach:

- MyFirstSessionBean is a POJO that exposes a (POJI), in this case MyBusinessInterface. This interface will be available to clients to invoke the EJB business methods.
- The **@Stateless** annotation indicates to the container that the given bean is stateless session bean so that the proper life cycle and runtime semantics can be enforced.
- By default, this session bean is accessed through a local interface.

This is all you need to set up a session EJB! There are no special classes to extend and no interfaces to implement. The very simple model of EJB 3.0 is shown in Figure 6-5.

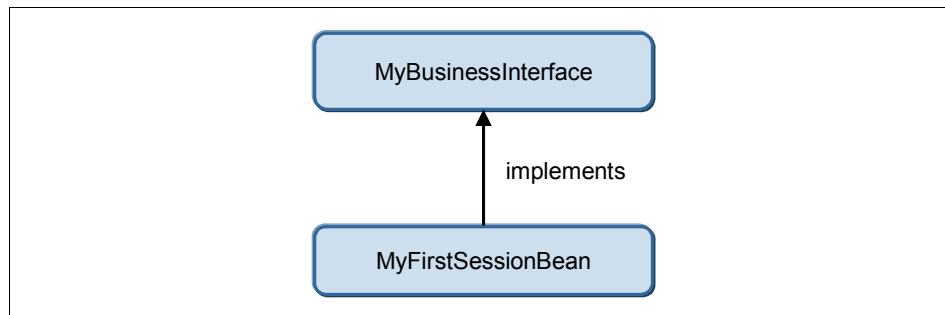


Figure 6-5 EJB is a POJO exposing a POJI

On the other hand, if we want to expose the same bean on the remote interface, we would use the **@Remote** annotation:

```
@Remote(MyRemoteBusinessInterface.class)
@Stateless
```

```

public class MyBean implements MyRemoteBusinessInterface {

    // ejb methods
    .....
}

```

**Tip:** If the session bean implements only one interface, you can also just code `@Remote` (without a class name).

## Stateful session EJBs

Stateful session EJBs are usually used to model a task or business process that spans multiple client requests, therefore, a stateful session bean retains state on behalf of an individual client. The client, on the other hand, has to store the handle to the stateful EJB, so that it always accesses the same EJB instance.

Using the same approach we adopted before, to define a stateful session EJB is to declare a POJO with the `@Stateful` annotation:

```

@Stateful
public class MySecondSessionBean implements MyBusinessStatefulInterface {

    // ejb methods
    .....
}

```

The **@Stateful** annotation indicates to the container that the given bean is stateful session bean so that the proper life cycle and runtime semantics can be enforced.

## Business interfaces

EJBs can expose different business interfaces, according to the fact that the EJB could be accessed either from a local or remote client. We recommend that common behaviors to both local and remote interfaces should be placed in a super-interface, as shown in Figure 6-6 on page 210.

Please take care of the following aspects:

- ▶ A business interface cannot be both a local and a remote business interface of the bean.
- ▶ If a bean class implements a single interface, that interface is assumed to be the business interface of the bean. This business interface will be a *local* interface, unless the interface is designated as a remote business interface by use of the `@Remote` annotation or by means of the deployment descriptor.

This gives you a great flexibility during the design phase, because you can decide which methods are visible to local and remote clients.

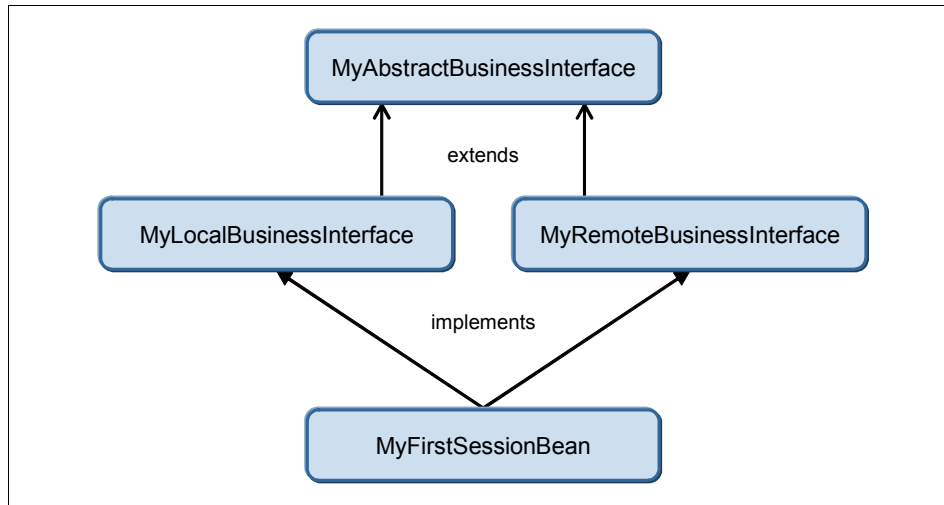


Figure 6-6 How to organize the EJB component interface

Using these guidelines, our first EJB is refactored as shown here:

```

@Stateless
public class MyFirstSessionBean
    implements MyLocalBusinessInterface, MyRemoteBusinessInterface {

    // implementation of methods declared in MyLocalBusinessInterface
    ....
    ....

    // implementation of methods declared in MyRemoteBusinessInterface
    ....
    ....
}
  
```

The MyLocalBusinessInterface is declared as interfaces with a @Local or @Remote annotation:

```

@Local
public interface MyLocalBusinessInterface
    extends MyAbstractBusinessInterface {
    // methods declared in MyLocalBusinessInterface
    .....
}

@Remote
public interface MyRemoteBusinessInterface
    extends MyAbstractBusinessInterface {
    // methods declared in MyRemoteBusinessInterface
  
```

```

        .....
    }

```

Another techniques to define the business interfaces exposed either as local or remote is to specify **@Local** or **@Remote** annotations with the full class that implements these interfaces:

```

@Stateless
@Local(MyLocalBusinessInterface.class)
@Remote(MyRemoteBusinessInterface.class)
public class MyFirstSessionBean implements MyLocalBusinessInterface,
                                           MyRemoteBusinessInterface {

    // implementation of methods declared in MyLocalBusinessInterface
    ....
    ....

    // implementation of methods declared in MyRemoteBusinessInterface
    ....
    ....
}

```

You can declare arbitrary exceptions on the business interface, but take into account the following rules:

- ▶ Do not use `RemoteException`.
- ▶ Any runtime exception thrown by the container is wrapped into an `EJBException`.

## 6.1.4 Best practices for developing EJBs

An EJB 3.0 developer must be adherent to the following basic rules:

- ▶ Each session bean must be a POJO, the class must be concrete (therefore neither abstract or final), and must have a no-argument constructor (if not present, the compiler will insert a default constructor).
- ▶ The POJO must implement *at least* one POJI. We stress at least, because you can have different interfaces for local and remote clients.
- ▶ If the business interface is `@Remote` annotated, all the values passed through the interface must implement `java.io.Serializable`. Typically the declared parameters are defined serializable, but this is not required as long as the actual values passed are serializable.

## 6.1.5 Message-driven beans

Exposing EJB 3.0 beans as message-driven beans (MDBs) using the `@MessageDriven` annotation is covered in Chapter 13, “Create the message-driven bean” on page 477, where we show how to implement an MDB that makes calls to EJB 3.0 session beans.

## 6.1.6 Web services

Exposing EJB 3.0 beans as Web services using the `@WebService` annotation is covered in Chapter 11, “Create the Web service” on page 409, where we show how to implement a Web service from an EJB 3.0 session bean.

## 6.1.7 Life cycle events

Another very powerful use of annotations is to *mark* callback methods for session bean life cycle events.

EJB 2.1 and prior releases required implementation of several life cycle methods, such as `ejbPassivate`, `ejbActivate`, `ejbLoad`, and `ejbStore`, for every EJB, even if you do not need these methods.

Because we use POJOs in EJB 3.0, the implementation of these life cycle methods has been made optional. Only if you implement any callback method in the EJB, would the container invoke that specific method.

The life cycle of a session bean can be categorized into several phases or events. The most obvious two events of a bean life cycle are creation and destruction for stateless session beans.

After the container creates an instance of a session bean, the container performs any *dependency injection* (described in the section that follows), and then invokes the method annotated with `@PostConstruct` (if there is one).

The client obtains a reference to a session bean, and then invokes a business method.

**Note:** The actual life cycle of a **stateless** session bean is independent of when a client obtains a reference to it. For example, the container can hand out a reference to the client, but not create the bean instance until some later time, for example, when a method is actually invoked on the reference. Or, the container can create a number of instances at startup time, and match them up with references at a later time.



At the end of the life cycle, the EJB container calls the method annotated with **@PreDestroy**, if there is one. The bean instance is then ready for garbage collection.

A stateless session bean with the two callback methods is shown here:

```
@Stateless
public class MyStatelessBean implements MyBusinessLogic {

    // .. bean business method

    @PostConstruct
    public void initialize() {
        // initialize the resources uses by the bean
    }

    @PreDestroy
    public void cleanup() {
        // deallocates the resources uses by the bean
    }
}
```

All stateless and stateful EJBs go through these two phases.

In addition, stateful session beans go through the passivation/activation cycle. Because an instance of a stateful bean is bound to a specific client (and therefore it cannot be reused among different requests), and the EJB container must manage the amount of physical available resources, the EJB container might decide to deactivate, or passivate, the bean by moving it from memory to secondary storage.

In correspondence with this more complex life cycle, we have further callback methods, specific to stateful session beans:

- ▶ The EJB container invokes the method annotated with **@PrePassivate**, immediately before passivating it.
- ▶ If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean by calling the method annotated with **@PostActivate**, if any, and then moves it to the ready stage.
- ▶ At the end of the life cycle, the client explicitly invokes a method annotated with **@Remove**, and the EJB container in turns calls the callback method annotated **@PreDestroy**.

**Note:** Because a stateful bean is bound to a particular client, it is best practice to correctly design session stateful beans to minimize their footprint inside the EJB container, and to correctly un-allocate it at the end of its life cycle, by invoking the method annotated with `@Remove`.

Stateful session beans have a *time-out* value. If the stateful session bean has not been used in the time-out period, it is marked inactive and is eligible for automatic deletion by the EJB container. Of course, it is still best practice for applications to remove the bean when the client is through with it, rather than relying on the time-out mechanism to do this.

Developers can explicitly invoke only the life cycle method annotated with `@Remove`, the other methods are invoked automatically by the EJB container.

## 6.1.8 Interceptors

The EJB 3.0 specification defines the ability to apply custom made interceptors to the business methods of both session and message driven beans. Interceptors take the form of methods annotated with the `@AroundInvoke` annotation (Example 6-1).

*Example 6-1 Applying an interceptor*

---

```
@Stateless
public class MySessionBean implements MyBusinessInterface {

    @Interceptors(LoggerInterceptor.class)
    public Customer getCustomer(String ssn) {
        ...
    }
    .....
}

public class LoggerInterceptor {
    @AroundInvoke
    public Object logMethodEntry(InvocationContext invocationContext)
                                throws Exception {
        System.out.println("Entering method: "
                           + invocationContext.getMethod().getName());
        Object result = invocationContext.proceed();
        // could have more logic here
        return result;
    }
}
```

---

We have the following notes for this example:

- ▶ The `@Interceptors` annotation is used to identify the session bean method where the interceptor should be applied;
- ▶ The `LoggerInterceptor` interceptor class defines a method (`logMethodEntry`) annotated with `@AroundInvoke`.
- ▶ The `logMethodEntry` method contains the advisor logic (in this case very simply logs the invoked method name), and invokes the `proceed` method on the `InvocationContext` interface to advice the container to proceed with the execution of the business method.

The implementation of interceptor in EJB 3.0 is a bit different from the analogous implementation of this aspect-oriented programming (AOP) paradigm that you can find in frameworks such as Spring or AspectJ, because EJB 3.0 does not support *before* or *after* advisors, but only *around* interceptors. However, *around* interceptors can act as *before* or *after* interceptors, or both. Interceptor code before the `invocationContext.proceed` call is run before the EJB method, and interceptor code after that call is run after the EJB method.

A very common use of interceptors is to provide preliminary checks (validation, security, and so forth) before the invocation of business logic tasks, and therefore they can throw exceptions. Because the interceptor is called together with the session bean code at run-time, these potential exceptions are sent directly to the invoking client.

In this sample we have seen an interceptor applied on a specific method; actually, the `@Interceptors` annotation can be applied at class level. In this case the interceptor will be called for every method.

Furthermore, the `@Interceptors` annotation accepts a list of classes, so that multiple interceptors can be applied to the same object.

**Note:** To give further flexibility, EJB 3.0 introduces the concept of a default interceptor that can be applied on every session (or MDB) bean contained inside the same EJB module. A default interceptor cannot be specified using an annotation, instead you should define it inside the deployment descriptor of the EJB module.

Note the following execution order in which interceptors are run:

- ▶ Default interceptor
- ▶ Class interceptors
- ▶ Method interceptors

To disable the invocation of a default interceptor or a class interceptor on a specific method, you can use the `@ExcludeDefaultInterceptors` and `@ExcludeClassInterceptors` annotations, respectively.

## 6.1.9 Dependency injection

The new specification introduces a powerful mechanism for obtaining Java EE resources (JDBC data source, JMS factories and queues, EJB references) and to inject them into EJBs, entities, or EJB clients.

In EJB 2.x the only way to obtain these resources was to use JNDI lookup using resource references, with a piece of code that could become cumbersome and vendor specific, because very often you had to specify properties related to the specific Java EE container provider.

EJB 3.0 adopts a *dependency injection* (DI) pattern, which is one of best ways to implement loosely coupled applications. It is much easier to use and more elegant than older approaches, such as dependency lookup through JNDI or container callbacks.

The implementation of dependency injection in the EJB 3.0 specification is based on annotations or XML descriptor entries, which allow you to inject dependencies on fields or setter methods.

Instead of complicated XML EJB references or resource references, you can use the `@EJB` and `@Resource` annotations to set the value of a field or to call a setter method within the beans with anything registered within JNDI. With these annotations, you can inject EJB references and resource references such as data sources and JMS factories.

In this section we show the most common usage of dependency injection in EJB 3.0.

### **@EJB annotation**

The `@EJB` annotation is used for injecting session beans into a client. This injection is only possible within managed environments, such as another EJB, or a servlet. We cannot inject an EJB into a JSF managed bean or Struts action, for example.

The parameters for the `@EJB` annotation are optional. The annotation parameters are:

- **name:** Specifies the JNDI name that is used to bind the injected EJB in the environment naming context (`java:comp/env`).

- ▶ **beanInterface:** Specifies the business interface to be used to access the EJB. By default, the business interface to be used is taken from the Java type of the field into which the EJB is injected. However, if the field is a supertype of the business interface, or if method-based injection is used rather than field-based injection, the **beanInterface** parameter is typically required, because the specific interface type to be used might be ambiguous without the additional information provided by this parameter.
- ▶ **beanName:** Specifies a *hint* to the system of the **ejb-name** of the target EJB that should be injected. It is analogous to the `<ejb-link>` stanza that can be added to an `<ejb-ref>` or `<ejb-local-ref>` stanza in the XML descriptor.

To access a session bean from a Java servlet we use the code shown in Example 6-2.

*Example 6-2 Injecting an EJB reference inside a servlet*

---

```
import javax.ejb.EJB;

public class TestServlet extends javax.servlet.http.HttpServlet
    implements javax.servlet.Servlet {

    // inject the remote business interface
    @EJB(beanInterface=MyRemoteBusinessInterface.class)
    MyAbstractBusinessInterface serviceProvider;

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        // call ejb method
        serviceProvider.myBusinessMethod();
        .....
    }
}
```

---

Here are some remarks on this example:

- ▶ We have specified the **beanInterface** attribute, because the EJB exposes two business interfaces (**MyRemoteBusinessInterface** and **MyLocalBusinessInterface**).
- ▶ If the EJB exposes only one interface, you are not required to specify this attribute, however, it can be useful to make the client code more readable.

## **@Resource annotation**

The **@Resource** annotation is the main annotation that can be used to inject resources in a managed component. In the following we show the most common usage scenarios of this annotation.

We show here how to inject a typical resource such as a data source inside a session bean.

### ***Field injection technique***

Example 6-3 on page 218 shows how to inject a data source inside a property that is used in a business method:

```
@Resource (name="jdbc/dataSource")
```

#### *Example 6-3 Field injection technique for a data source*

---

```
@Stateless
public class CustomerSessionBean implements CustomerServiceInterface {

    @Resource (name="jdbc/dataSource")
    private DataSource ds;

    public void businessMethod1() {
        java.sql.Connection c=null;
        try {
            c = ds.getConnection();
            // .. use the connection
        } catch (java.sql.SQLException e) {
            // ... manage the exception
        } finally {
            // close the connection
            if(c!=null) {
                try { c.close(); } catch (SQLException e) { }
            }
        }
    }
}
```

---

All parameters for the `@Resource` annotation are optional. The annotation parameters are:

- ▶ **name**: Specifies the component-specific internal name—the resource reference name—within the **java:comp/env** name space. It does not specify the global JNDI name of the resource being injected. A binding of the reference to a JNDI name is necessary to provide that linkage, just as it is in J2EE 1.4.
- ▶ **type**: Specifies the resource manager connection factory type.
- ▶ **authenticationType**: Specifies whether the container or the bean is to perform authentication.
- ▶ **shareable**: Specifies whether resource connections are shareable or not.

- ▶ `mappedName`: Specifies a product specific name that the resource should be mapped to. WebSphere does not make any use of `mappedName`.
- ▶ `description`: Description.

### ***Setter method injection***

Another technique is to inject a setter method. The setter injection technique is based on JavaBeans property naming conventions (Example 6-4 on page 219).

*Example 6-4 Setter method injection technique for a data source*

---

```
@Stateless
public class CustomerSessionBean implements CustomerServiceInterface {

    private Datasource ds;

    @Resource (name="jdbc/dataSource")
    public void setDatasource(DataSource datasource) {
        this.ds = datasource;
    }
    ...
    public void businessMethod1() {
        ...
    }
}
```

---

Here are some remarks on these two examples:

- ▶ In this example, we directly used the data source inside the session bean. This is not good practice, because you should put JDBC code in specific components, such as data access objects.
- ▶ We recommend that you use the setter injection technique, which gives more flexibility:
  - You can put initialization code inside the setter method.
  - The session bean can be easily tested as a stand-alone component.

Other interesting usages of `@Resource` are as follows:

- ▶ To obtain a reference to the EJB session context:

```
@Stateless
public class CustomerSessionBean implements CustomerServiceInterface {
    ....
    @Resource javax.ejb.SessionContext ctx;
}
```

- ▶ To obtain the value of an environment variable, which is configured inside the deployment descriptor with env-entry:

```
@Stateless
public class CustomerSessionBean implements CustomerServiceInterface {
    ....
    @Resource String myEnvironmentVariable;
}
```

- ▶ Injection of JMS resources, such as JMS factories or queues.

### 6.1.10 Using deployment descriptors

Up to now we have seen how to define an EJB, how to inject resources into it, or how to specify its life cycle event with annotations. We can get the same result by specifying a deployment descriptor (`ejb-jar.xml`) with the necessary information in the EJB module.

### 6.1.11 EJB 3.0 application packaging

Session beans and MDBs can be packaged in Java standard JAR file. This can be achieved by two strategies:

- ▶ Using a Java project or Java Utility project
- ▶ Using an EJB project

If you use the first approach you have to add the Java Project to the EAR project, by editing the deployment descriptor (`application.xml`), and adding the lines:

```
<module>
  <ejb>MyEJB3Module.jar</ejb>
</module>
```

When using the second approach, the workbench can automatically update the `application.xml` file, and set up an `ejb-jar.xml` inside the EJB project. However, in EJB 3.0 you are not required to define the EJBs and related resources in an `ejb-jar.xml` file, because they are usually defined through the use of annotations. The main usage of deployment descriptor files is to override or complete behavior specified by annotations.

## 6.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.



For this chapter to work, make sure that you have completed these steps:

- ▶ Defined the two databases (Vacation and Donate)
- ▶ Defined the matching JDBC data sources (jdbc/Vacation and jdbc/Donate)
- ▶ Created the JPA entities (Employee and Fund)
- ▶ Created and run the JUnit test case (EntityTester)

## 6.3 Create and configure an EJB 3.0 project

In this section, we create the DonateEJB project to be contained in the DonateEAR project. An EJB3.0 project generally contains session beans, message-driven beans, and JPA entities, all packaged together as a JAR file.

In our design, we created the JPA entities in separate JPA projects, DonateJPAEmployee and DonateJPAFund. The DonateEJB project only contains session beans and a message-driven bean.

### 6.3.1 Create the Project

1. In the Java EE perspective Enterprise Explorer, select anywhere in the whitespace, right-click and select **New** → **EJB Project**.
2. At the New EJB Project/EJB Project pop-up:
  - Set the project name as **DonateEJB**.
  - Accept the default of 3.0 for EJB Module version.
  - a. Set configuration to **Default Configuration for WebSphere Application Server v7.0**
  - b. Enable **Add project to an EAR** and set EAR Project Name to **DonateEAR**.
  - c. Click **Next**.

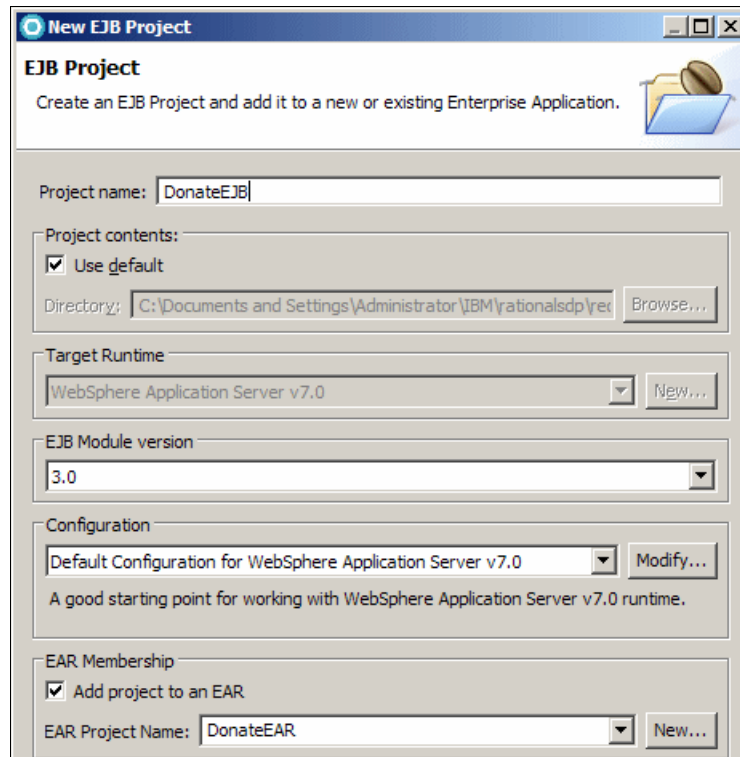


Figure 6-7 EJB project settings

3. At the New EJB Project/EJB Module pop-up:
  - a. Clear **Create an EJB Client JAR module to hold the client interfaces and classes**
  - b. Enable **Generate Deployment Descriptor**.
  - c. Click **Finish**.

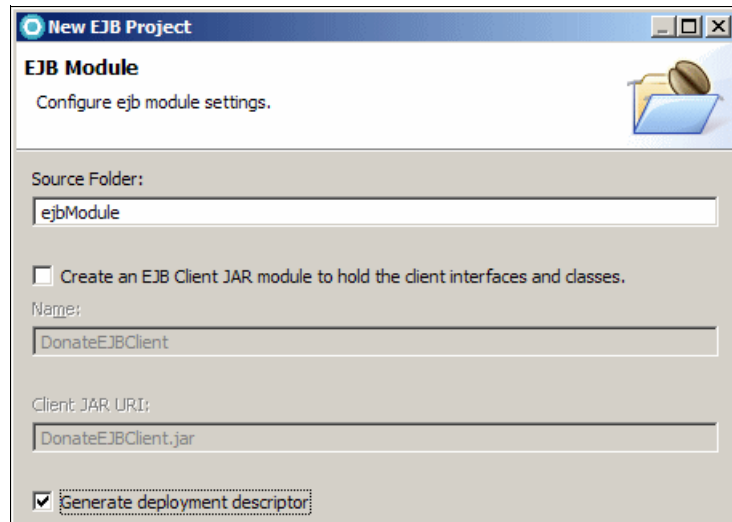


Figure 6-8 EJB module settings

4. Optionally review and then close the Technology Quickstarts editor that opened after DonateEJB was created. These quickstarts are provided by Rational Application Developer to assist the novice developer learn more about the technology artifact that was just created.

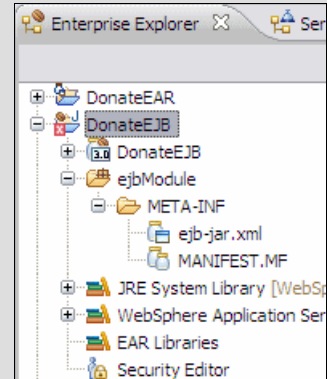


Figure 6-9 Technology Quickstarts page

### Behind the scenes:

- ▶ .DonateEJB is now visible in the Enterprise Explorer with an EJB deployment descriptor (ejb-jar.xml) in ejbModule/META-INF
- ▶ The error in DonateEAR has been resolved because DonateEJB has been added to application.xml (in DonateEAR/META-INF)
- ▶ DonateEJB contains the following error:
  - An EJB module must contain one or more enterprise beans.

This error will be resolved in 6.4, “Create the entity facade interfaces and session beans” on page 225



## 6.3.2 Add project dependencies

The DonateEJB project requires access to the JPA entities to interact with the databases.

1. Select **DonateEJB**, right-click and select **Properties**.
2. At the Properties pop-up:
  - a. On the left select **Java EE Module Dependencies**,
  - b. On the right select the two JPA projects, **DonateJPAEmployee.jar** and **DonateJPAFund.jar**.
  - c. Click **OK**.

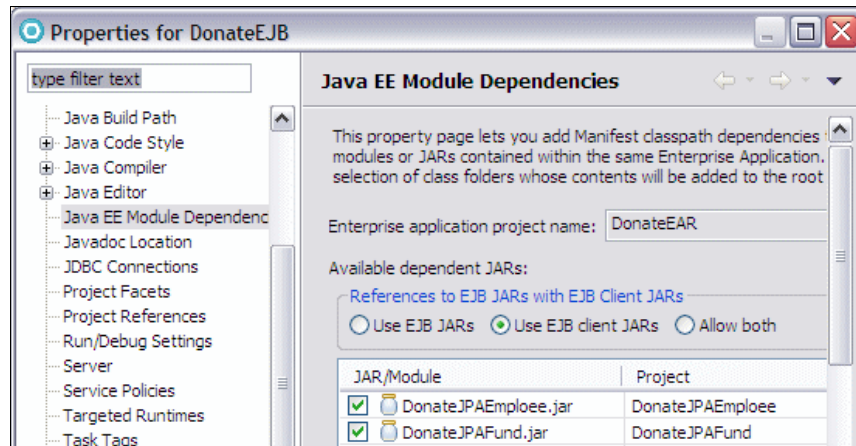


Figure 6-10 Java EE module dependencies

## 6.4 Create the entity facade interfaces and session beans

We do not want clients (such as servlets) to use the JPA entities directly. Therefore, we provide a session facade for each entity, and a business session bean for general application logic.

In this section we create the session bean facade for the Employee and the Fund entities.

### 6.4.1 Define the business interface for the employee facade

First we define the business interface that is implemented by the employee facade session bean.

1. In the Enterprise Explorer, select **DonateEJB**, right-click and select **New** → **Interface**.
2. At the New Java Interface/Java Interface pop-up:
  - a. Set Package to **vacation.ejb.interfaces**.
  - b. Set name to **EmployeeFacadeInterface**.
  - c. Click **Finish**.

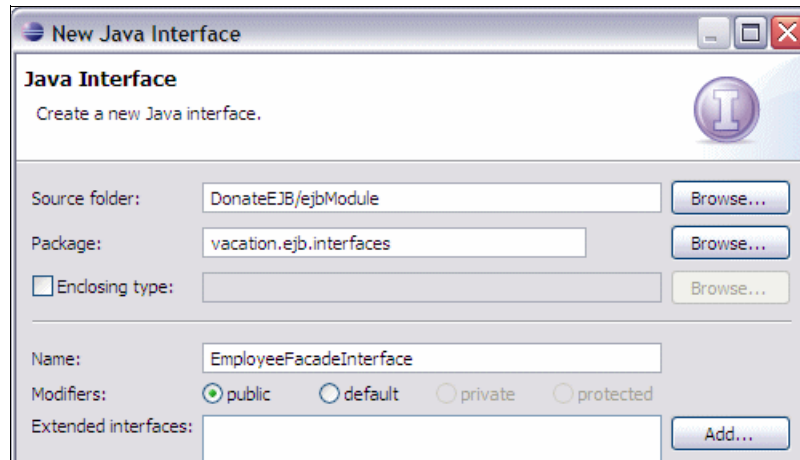


Figure 6-11 Interface properties

3. In the resulting EmployeeFacadeInterface.java editor:
  - a. Move the cursor to the interface body (before the closing brace in the file) and insert the **F06.1 EmployeeFacade Interface** snippet from the Snippets view.
 

```
public interface EmployeeFacadeInterface {
    public Employee findEmployeeById(int id) throws Exception;
    public Employee addVacationHours(Employee employee, int hours);
    public Employee deductVacationHours(Employee employee, int hours)
                                   throws Exception;
}
```
  - b. Organize imports (**Ctrl+Shift+O**) to resolve vacation.entities.Employee.
  - c. Save and close EmployeeFacadeInterface.java.

#### Behind the scenes:

The business interface, by default, is a local interface. We could add the `@Local` annotation to the interface definition to explicitly state that this interface is intended to be used as a local interface, or we could define a separate local interface that extends the business interface.

In the section that follows, we define a remote interface that extends the business interface.

## 6.4.2 Define the remote interface for the employee facade

The session bean will be accessed from remote clients. Therefore, we also define a remote interface for the session bean facade:

1. In the Enterprise Explorer, select **vacation.ejb.interfaces** (in DonateEJB/ejbModule), right-click and select **New** → **Interface**.
2. At the New Java Interface pop-up,
  - a. Set Name to **EmployeeFacadeRemote**.
  - b. Next to Extended Interfaces, click **Add**.
    - At the Extended Interfaces Selection pop-up, begin typing **EmployeeFacadeInterface** until it appears in the Matching item list. Select it from Matching items and click **OK**.

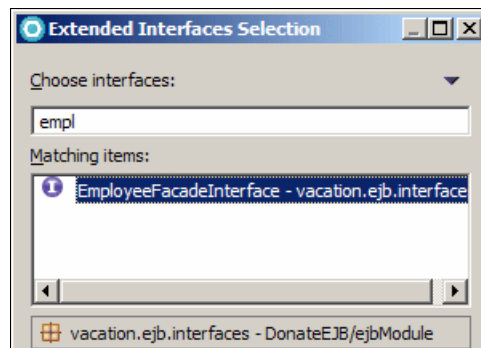


Figure 6-12 Select the interface

Back at the New Java Interfaces pop-up, note that EmployeeFacadeInterface is now listed in the Extended interfaces list.

- c. Click **Finish**.

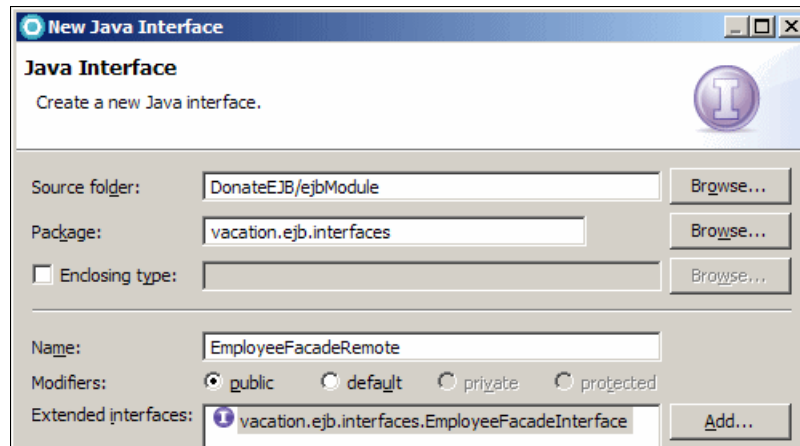


Figure 6-13 Extended interface has been added

3. In the resulting EmployeeFacadeRemote.java editor:
  - a. Add a **@Remote** annotation above the interface signature:
 

```
@Remote
public interface EmployeeFacadeRemote extends EmployeeFacadeInterface {
```

**Behind the scenes:** The @Remote annotation makes this interface available to clients that are running outside of the EJB container where this EJB is running.

- b. Organize imports (**Ctrl+Shift+O**) to resolve javax.ejb.Remote.
  - c. Save and close EmployeeFacadeRemote.java.

### 6.4.3 Create the session bean facade for the employee entity

In this section we create the session bean facade (EmployeeFacade) for the Employee entity:

1. In the Enterprise Explorer, select **DonateEJB**, right-click and select **New** → **Session Bean**.
2. At the Create EJB 3.0 Session Bean pop-up:
  - a. Set Java package to **vacation.ejb.impl**.
  - b. Set Class name to **EmployeeFacade**.
  - c. Clear the **Create business interface** check boxes.



**Behind the scenes:** you clear these selections because you created the local and remote interfaces in the previous section.

- d. Click **Next**.

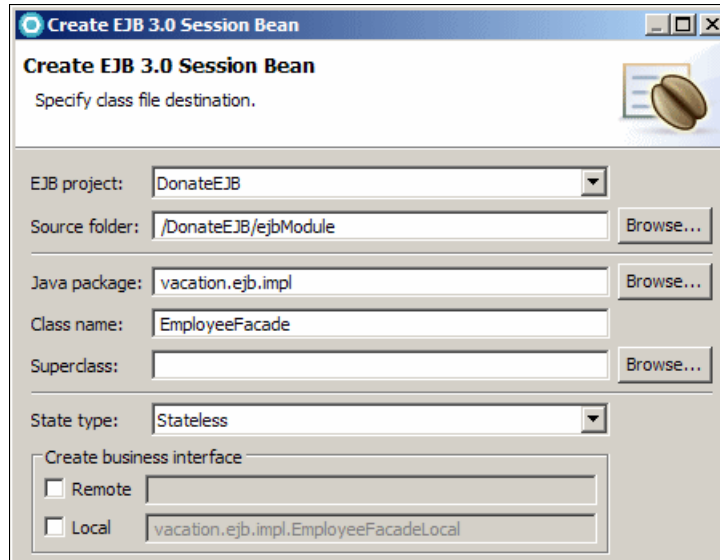


Figure 6-14 Create the session bean

3. At the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, click **Add**.
  - a. At the Choose Interface pop-up, begin typing **EmployeeFacadeInterface** until it appears in the Matching item list. Select it from Matching items
  - b. Under Add as select **Local**.
  - c. Click **OK**.

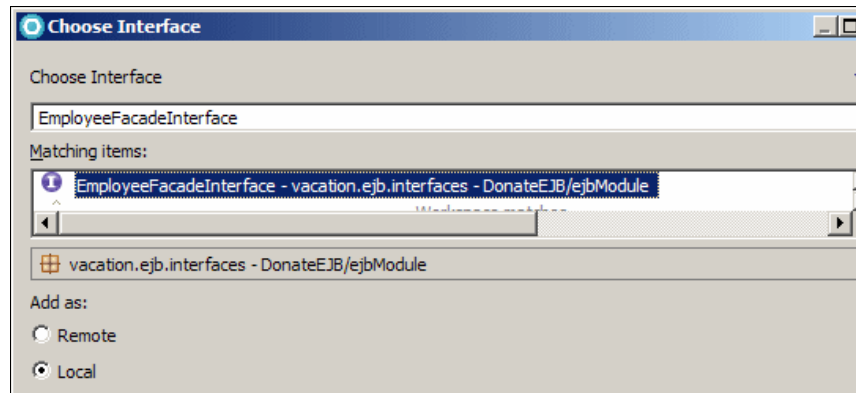


Figure 6-15 Select the interface

4. Back at the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, click **Add**.
  - a. At the Choose Interface pop-up, begin typing **EmployeeFacadeRemote** until it appears in the Matching item list. Select it from Matching items
  - b. Under Add as select **Remote**.
  - c. Click **OK**
5. Back at the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, note that both the local and remote interface appears, and click **Next**.

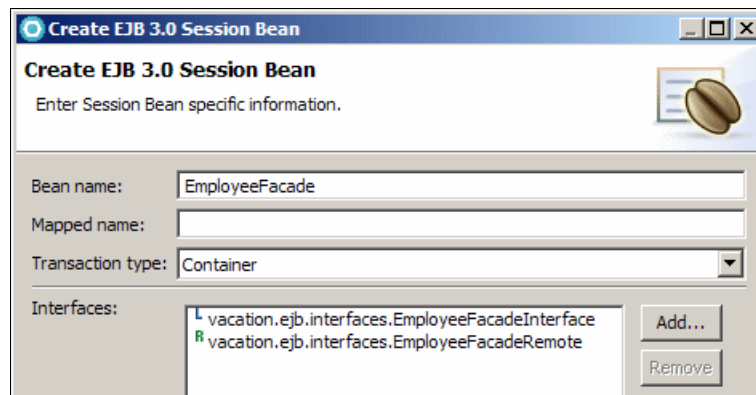


Figure 6-16 Session bean's local and remote interfaces

6. At the Create EJB 3.0 Session Bean/Select Class Diagram for Visualization:
  - a. Clear **Add bean to Class Diagram**.

- b. Click **Finish**.

**Behind the scenes:** Rational Application Developer class diagrams provide a visual representation of EJBs. Further information is available in the Rational Application Developer InfoCenter:

<http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.xtools.viz.ejb.doc/topics/teditejbcompnt.html>

7. The `EmployeeFacade.java` editor opens automatically with the generated skeleton session bean (Example 6-5).

*Example 6-5 Skeleton employee facade session bean*

---

```
package vacation.ejb.impl;

import vacation.ejb.interfaces.EmployeeFacadeInterface;
import vacation.ejb.interfaces.EmployeeFacadeRemote;
import vacation.entities.Employee;

@Stateless
@Local( { EmployeeFacadeInterface.class })
@Remote( { EmployeeFacadeRemote.class })
public class EmployeeFacade implements EmployeeFacadeInterface,
EmployeeFacadeRemote {

    public Employee addVacationHours(Employee employee, int hours) {
        // TODO Auto-generated method stub
        return null;
    }

    public Employee deductVacationHours(Employee employee, int hours)
        throws Exception {
        // TODO Auto-generated method stub
        return null;
    }

    public Employee findEmployeeById(int id) throws Exception {
        // TODO Auto-generated method stub
        return null;
    }
}
```

---

### Behind the scenes:

Note that the three methods skeletons were created from the three method signatures in the `EmployeeFacadeInterface` (`addVacationHours`, `deductVacationHours`, and `findEmployeeById`).

**Behind the scenes:** The `@Stateless` annotation declares this Java class as a stateless session EJB

8. Add an `EntityManager` to this class by adding the following code to the class body, directly above the method signatures

- a. Organize imports (**Ctrl+Shift+O**). If prompted to specify which `EntityManager` class to use, select **`javax.persistence.EntityManager`**

```
public class EmployeeFacade implements EmployeeFacadeInterface,
                                         EmployeeFacadeRemote {

    @PersistenceContext(unitName = "DonateJPAEmployee")
    private EntityManager em;

    public Employee addVacationHours(Employee employee, int hours) {
        ....
    }
}
```

**Behind the scenes:** The `EntityManager` will coordinate accessing information associated with the `DonateJPAEmployee` persistence unit.

Recall that we defined the `DonateJPAEmployee` persistence unit in the `persistence.xml` file in 5.7.5, “Add the Fund entity to the `persistence.xml`” on page 178.

9. Delete the generated skeleton methods. Note that the generated comments lines were omitted from the deleted fragment shown below.

```
public Employee addVacationHours(Employee employee, int hours) {
    // TODO Auto-generated method stub
    return null;
}

public Employee deductVacationHours(Employee employee, int hours) {
    // TODO Auto-generated method stub
    return null;
}

public Employee findEmployeeById(int id) {
    // TODO Auto-generated method stub
}
```

```

        return null;
    }

```

10. Before the closing brace, insert the **F06.2 Employee Facade** snippet from the Snippets view. Refer to the following **Behind the scenes** box for details about the inserted methods.
11. Save and close `EmployeeFacade.java`.

### Behind the scenes:

The **F06.2 Employee Facade** snippet inserted completed method definitions for the three required methods (from `EmployeeFacadeInterface`) along with a common helper method:

- An `addHours` helper method is defined:

```

public Employee addHours(Employee employee, int hours) {
    employee = em.find(Employee.class, employee.getEmployeeId());
    employee.setVacationHours(employee.getVacationHours() + hours);
    em.merge(employee);
    return employee;
}

```

This method retrieves an employee and adds the specified number of hours to the `vacationHours` property.

The employee object that is passed into this method could have been created outside of this transaction context and thus might be stale, that is containing an older version of the employee record. Therefore, this method transactionally retrieves the current employee element and updates only the `vacationHours` field.

- The `addVacationHours` method calls the `addHours` method:

```

public Employee addVacationHours(Employee employee, int hours) {
    return addHours((Employee) employee, hours);
}

```

- The `deductVacationHours` method verifies that enough hours are available before subtracting the specified number of hours (using the `addHours` method):

```

public Employee deductVacationHours(Employee employee, int hours)
    throws Exception {
    if (employee.getVacationHours() < hours)
        throw new Exception
            ("Employee does not have sufficient hours to deduct
from");
    return addHours(employee, -hours);
}

```

- The `findEmployeeById` method retrieves the employee and throws an exception if the employee is not found:

```
public Employee findEmployeeById(int id) throws Exception {
    Employee employee = em.find(Employee.class, id);
    if (employee == null)
        throw new Exception("Employee with id " + id + " not
found");
    return employee;
}
```

#### 6.4.4 Define the business interface for the fund facade

We define the business interface that is implemented by the fund facade session bean in the same way as for the employee facade (see 6.4.1, “Define the business interface for the employee facade” on page 225):

1. In the Enterprise Explorer, select **DonateEJB**, right-click and select **New** → **Interface**.
2. At the New Java Interface/Java Interface pop-up:
  - a. Set Package to **donate.ejb.interfaces**.
  - b. Set name to **FundFacadeInterface**.
  - c. Click **Finish**.
3. In the resulting `FundFacadeInterface.java` editor, move the cursor to the interface body (before the closing brace in the file) and insert the **F06.3 FundFacade Interface** snippet from the Snippets view.

```
public interface FundFacadeInterface {
    public abstract List<Fund> getAllFunds();
    public abstract Fund findFundByName(String name) throws Exception;
    public abstract Fund createNewFund(String name) throws Exception;
    public void removeFund(Fund fund);
    public abstract Fund addToBalance(Fund fund, int amount);
    public abstract Fund deductFromBalance(Fund fund, int amount)
        throws Exception;
}
```

4. Organize imports (**Ctrl+Shift+O**) to resolve:

```
import java.util.List;
import donate.entities.Fund;
```

5. Save and close `FundFacadeInterface.java`.

## 6.4.5 Define the remote interface for the fund facade

For access by remote clients, we also define a remote interface for the session bean facade in the same way as for the remote employee facade (see 6.4.2, “Define the remote interface for the employee facade” on page 227):

1. In the Enterprise Explorer, select **donate.ejb.interfaces** (in `DonateEJB/ejbModule`), right-click and select **New** → **Interface**.
2. At the New Java Interface pop-up,
  - a. Set Name to **FundFacadeRemote**.
  - b. Next to Extended Interfaces, and click **Add**.
    - i. At the Extended Interfaces Selection pop-up, begin typing **FundFacadeInterface** until it appears in the Matching item list. Select it from Matching items and click **OK**.
  - c. Back at the New Java Interfaces pop-up, note that `FundFacadeInterface` is now listed in the Extended interfaces list.
  - d. Click **Finish**.
3. In the resulting `FundFacadeRemote.java` editor:
  - a. Add a **@Remote** annotation above the method signature:

```
@Remote
public interface FundFacadeRemote extends FundFacadeInterface {
```
  - b. Organize imports (**Ctrl+Shift+O**) to resolve `javax.ejb.Remote`.
  - c. Save and close `FundFacadeRemote.java`.

## 6.4.6 Create the session bean facade for the fund entity

In this section we create the session bean facade (`FundFacade`) for the `Fund` entity in the same way as for the `Employee` entity (see 6.4.3, “Create the session bean facade for the employee entity” on page 228):

1. In the Enterprise Explorer, select **DonateEJB**, right-click and select **New** → **Session Bean**.
2. At the Create EJB 3.0 Session Bean pop-up:
  - a. Set Java package to **donate.ejb.impl**.
  - b. Set Class name to **FundFacade**.
  - c. Clear the **Create business interface** check boxes.
  - d. Click **Next**.

3. At the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, click **Add**.
  - a. At the Choose Interface pop-up, begin typing **FundFacadeInterface** until it appears in the Matching item list. Select it from Matching items
  - b. Under Add as select **Local**.
  - c. Click **OK**.
4. Back at the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, click **Add**.
  - a. At the Choose Interface pop-up, begin typing **FundFacadeRemote** until it appears in the Matching item list. Select it from Matching items
  - b. Under Add as select **Remote**.
  - c. Click **OK**
5. Back at the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, note that both the local and remote interface appears, and click **Next**.

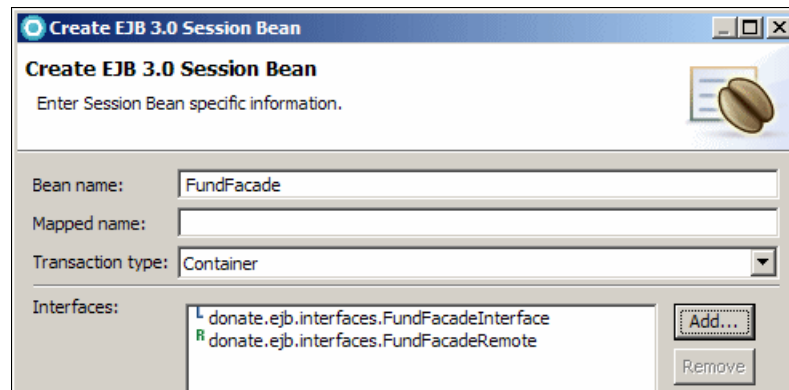


Figure 6-17 Session bean's local and remote interfaces

6. At the Create EJB 3.0 Session Bean/Select Class Diagram for Visualization:
  - a. Clear **Add bean to Class Diagram**.
  - b. Click **Finish**.
7. The FundFacade.java editor opens automatically with the generated skeleton session bean (Example 6-6).

*Example 6-6 Skeleton fund facade session bean*

---

```
package donate.ejb.impl;
```



```

import java.util.List;

import donate.ejb.interfaces.FundFacadeInterface;
import donate.ejb.interfaces.FundFacadeRemote;
import donate.entities.Fund;

@Stateless
@Local( { FundFacadeInterface.class })
@Remote( { FundFacadeRemote.class })
public class FundFacade implements FundFacadeInterface, FundFacadeRemote {
    public Fund addToBalance(Fund fund, int amount) {
        // TODO Auto-generated method stub
        return null;
    }

    public Fund createNewFund(String name) throws Exception {
        // TODO Auto-generated method stub
        return null;
    }

    public Fund deductFromBalance(Fund fund, int amount) throws Exception {
        // TODO Auto-generated method stub
        return null;
    }

    public Fund findFundByName(String name) throws Exception {
        // TODO Auto-generated method stub
        return null;
    }

    public List<Fund> getAllFunds() {
        // TODO Auto-generated method stub
        return null;
    }

    public void removeFund(Fund fund) {
        // TODO Auto-generated method stub
    }
}

```

### Behind the scenes:

Note that the six methods skeletons were created from the six method signatures in the FundFacadeInterface (createNewFund, deductFromBalance, findFundByName, getAllFunds, and removeFund).

8. Add an `EntityManager` to this class by adding the following code to the class body, directly above the method signatures:

- a. Organize imports (**Ctrl+Shift+O**). If prompted to specify which `EntityManager` class to use, select **javax.persistence.EntityManager**

```
@PersistenceContext(unitName = "DonateJPAFund")
private EntityManager em;

public FundFacade() {
    // TODO Auto-generated constructor stub
}
```

9. In the Java Editor, delete the generated skeleton methods, then insert the **F06.4 Fund Facade** snippet from the Snippets view before the closing brace. Refer to the following **Behind the scenes** box for details about the inserted methods.

10. Optionally add the annotation **@SuppressWarnings("unchecked")** before the `getAllFunds` method (this removes the warning).

```
@SuppressWarnings("unchecked")
public List<Fund> getAllFunds() {
    return (List<Fund>)em.createNamedQuery("getAllFunds").getResultList();
}
```

11. Save and close `FundFacade.java`.

#### Behind the scenes:

The **F06.4 Fund Facade** snippet inserted completed method definitions for the six required methods (from `FundFacadeInterface`) along with a common helper method:

- The `getAllFunds` method uses the `getAllFunds` named query of the `Fund` entity to retrieve all the funds:

```
public List<Fund> getAllFunds() {
    return (List<Fund>)em.createNamedQuery("getAllFunds")
                           .getResultList();
}
```

- The `findFundByName` method retrieves one fund and throws an exception if the fund is not found:

```
public Fund findFundByName(String name) throws Exception {
    Fund fund = em.find(Fund.class, name);
    if (fund == null)
        throw new Exception
            ("Fund not found with name \"" + name + "\"");
    return fund;
}
```

- The `createNewFund` method verifies that the fund does not exist already, then adds the new fund:

```
public Fund createNewFund(String name) throws Exception {  
    Fund fund = null;  
    try {  
        fund = findFundByName(name);  
    } catch(Exception e) { }  
    if (fund != null)  
        throw new Exception("Fund name already exists");  
    fund = new Fund();  
    fund.setName(name);  
    fund.setBalance(0);  
    em.persist(fund);  
    return fund;  
}
```

- The `removeFund` method removes a fund:

```
public void removeFund(Fund fund) {  
    fund = em.find(Fund.class, fund.getName());  
    em.remove(fund);  
}
```

- The `addToBalance` method adds donated hours to the fund:


```
public Fund addToBalance(Fund fund, int amount) {  
    fund = em.find(Fund.class, fund.getName());  
    fund.setBalance(fund.getBalance() + amount);  
    em.merge(fund);  
    return fund;  
}
```

- The `deductFromBalance` method subtracts hours from the fund if enough hours exist:

```
public Fund deductFromBalance(Fund fund, int amount)  
                                throws Exception {  
    if (fund.getBalance() < amount)  
        throw new Exception  
            ("Fund does not have sufficient amount to deduct from");  
    return addToBalance(fund, -amount);  
}
```

## 6.5 Deploy the DonateEAR enterprise application

As a preparation for testing with JUnit, we deploy the DonateEAR enterprise application to the server and verify that deployment succeeds:

1. Ensure that the test environment is running:
  - a. If the Derby Network Server is not running, open a command window, change directory to
    -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`and execute:
    -  `startNetworkServer.bat`
    -  `startNetworkServer.sh`
  - b. If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
    - If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.

### Behind the scenes:

Recall that you started both of these in earlier chapters. They should still be running unless you stopped them or restarted the system.

- ▶ 4.2, “Start the Derby Network Server” on page 87.
- ▶ 3.6, “Start the ExperienceJEE Test Server” on page 78.

2. In the Servers view, select **ExperienceJEE Test Server**, right-click and select **Add and Remove Projects**.
  - a. Select the **DonateEAR** project on the left and click **Add**.
  - b. Click **Finish** to deploy the enterprise application to the server.

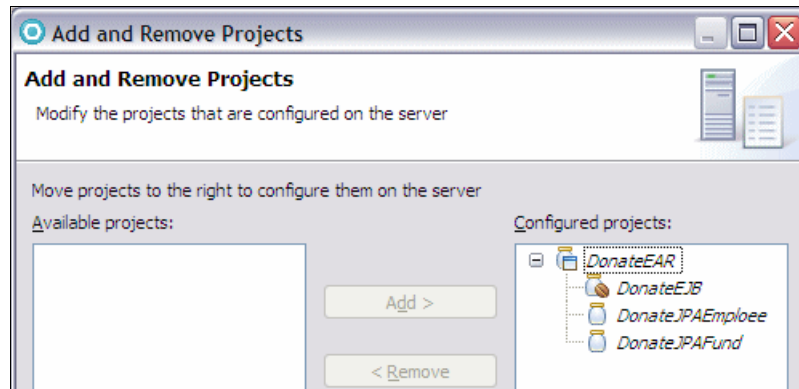


Figure 6-18 Add the projects to the server

**Behind the scenes:** Note that DonateEAR contains the DonateEJB, DonateJPAEmployee, and DonateJPAFund projects.

The project will be automatically published. Recall that you disabled automatic publishing to the server, but that only applies to publishing updates: applications are still published when they are initially added.

3. The application status will change from publishing
4. Switch to the Console view and verify that the application started. In particular look for the following two lines indicating that DonateEAR and DonateEJB both successfully started:
 

```
TWSVR0057I: EJB jar started: DonateEJB.jar
WSVR0221I: Application started: DonateEAR
```

#### Off Course?

- ▶ If you do not see the Console view, open it from the workbench action bar by selecting **Window** → **Show View** → **Console**.
  - The Console view will contain many messages and you may need to scroll up from the bottom to see these DonateEAR startup messages.

### Behind the scenes:

During server startup the server, as specified by the EJB 3.0, defines the bindings based on a standard name for the beans.

In the server logs you can see the following lines:

Starting EJB jar: DonateEJB.jar

The server is binding the donate.ejb.interfaces.FundFacadeRemote interface of the FundFacade enterprise bean in the DonateEJB.jar module of the DonateEAR application. The binding location is:  
ejb/DonateEAR/DonateEJB.jar/FundFacade#donate.ejb.interfaces.FundFacadeRemote

The server is binding the donate.ejb.interfaces.FundFacadeRemote interface of the FundFacade enterprise bean in the DonateEJB.jar module of the DonateEAR application. The binding location is:  
donate.ejb.interfaces.FundFacadeRemote

The same session bean is bound to two several JNDI entries. The default binding pattern is as follows:

Short form local interfaces and homes

- ejblocal:<package.qualified.interface>

Short form remote interfaces and homes

- <package.qualified.interface>

Long form local interfaces and homes

- ejblocal:<component-id>#<package.qualified.interface>

Long form remote interfaces and homes

- ejb/<component-id>#<package.qualified.interface>

where the <component-id> defaults to  
<application-name>/<module-jar-name>/<ejb-name> unless it is overridden in the EJB module binding file.

Refer to

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.zseries.doc/info/zseries/ae/cejb\\_bindingsejbfp.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.zseries.doc/info/zseries/ae/cejb_bindingsejbfp.html)

## 6.6 Test the session facades with the Universal Test Client

The Universal Test Client (UTC) is a test tool shipped with Rational Application Developer that allows you to perform basic testing of your Java EE artifacts without writing test code. For example, in this chapter you use the UTC to test entity EJBs. The UTC is launched as a window within the workbench or it can be accessed in a browser using:

<http://localhost:9080/UTC>

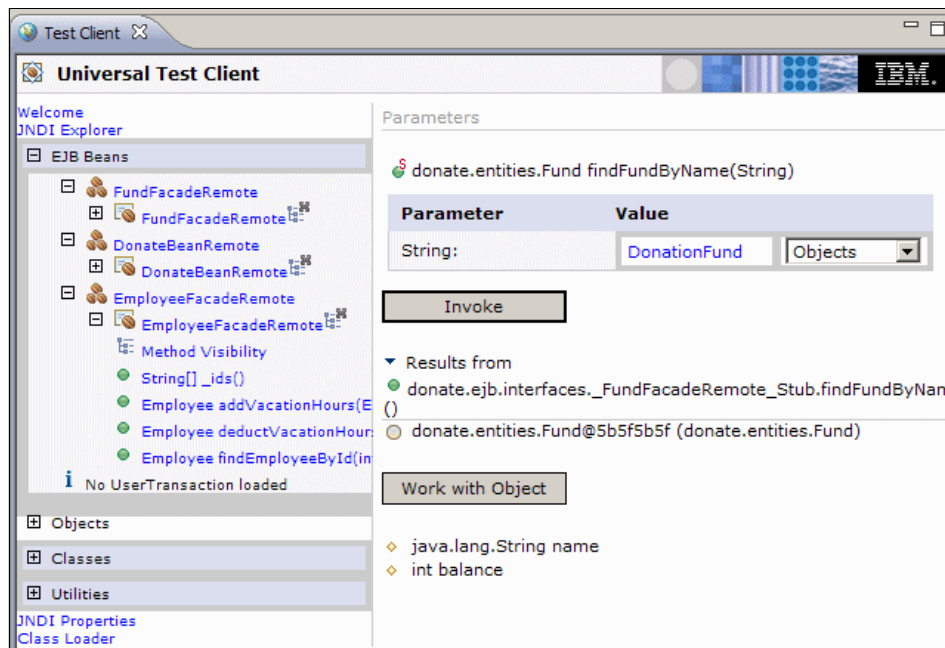


Figure 6-19 Universal test client

In the test client:

- ▶ On the left is the resource tree, which contains the various functional selections such as the JNDI tree, an EJB class (either a home interface or an entity EJB instance), or the Class Loader properties.
- ▶ On the right is the details pane, and the content depends on what was selected on the left:
  - If you select the JNDI explorer, this shows the list of available JNDI names and local EJB references. From this page, you can select an EJB home interface (local or remote), and it is added to the EJB tree on the left

- If you select a class from the EJB beans tree, this shows the available operation and results. In the above example, the setFundBalance method was selected from an entity EJB instance on the left.

If you select a properties selection (like the JNDI properties or the Class Loading) you can set properties such as the user ID and password

### 6.6.1 Start the UTC

1. Switch to the **Servers** view in the lower-right pane.
2. Select the **ExperienceJEE Test Server**, right-click and select **Universal test client** → **Run**.
3. At the resulting Test Client browser window, login using the **javaeeadmin** User ID and password that you specified in section 3.4, “Create the ExperienceJEE server profile” on page 62.

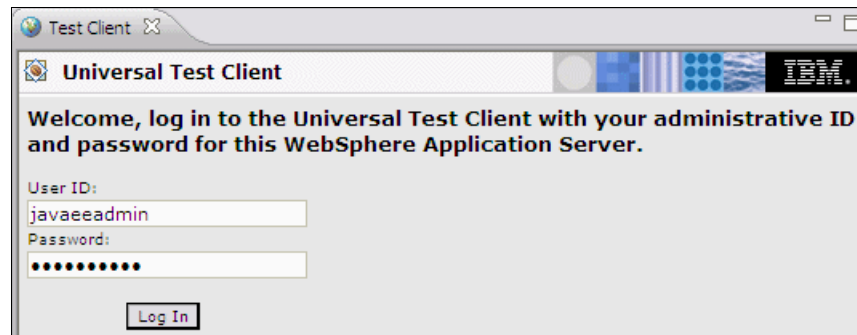


Figure 6-20 Log in to the test client

**Behind the scenes:** since administrative security is defined in the ExperienceJEE Test Server, the UTC requires the userid and password to query the classloader definitions and access the application class files.

### 6.6.2 Test the EmployeeFacade EJB with the UTC

1. In the **Test Client** pane on the upper left, select **JNDI Explorer**.



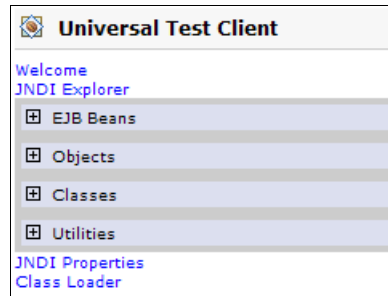


Figure 6-21 Select JNDI explorer

2. In the **JNDI Explorer** pane on the right, open the reference to the entity EJB local home interface by selecting **ejb** → **DonateEAR** → **DonateEJB.jar** → **EmployeeFacade#vacation.ejb.interfaces.EmployeeFacadeRemote**.

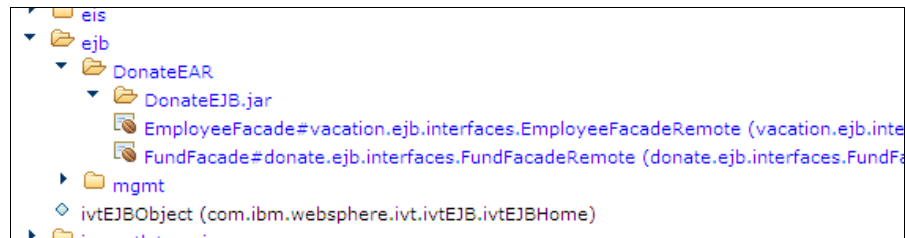


Figure 6-22 Open the reference to the entity EJB

3. In the **EJB Beans** section located on the left, open the desired entity EJB instance by selecting **EmployeeFacadeRemote** → **EmployeeFacadeRemote** → **Employee findEmployeeById(int)**.

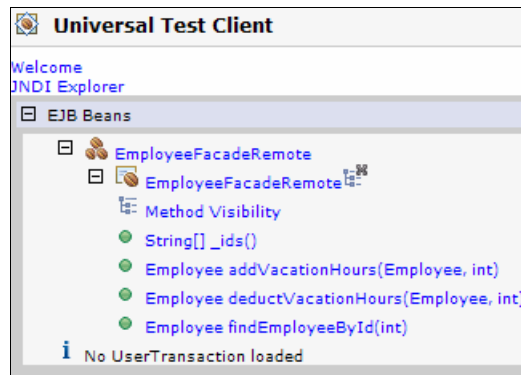


Figure 6-23 Open the EJB instance

4. In the resulting right pane, under the **Value** column enter **1** and click **Invoke**.  
The operation will complete with a message similar to Figure 6-24 in the lower right, indicating that the instance was located. Note that you may have to expand the collapsed Results section to see all the information
  - a. Click **Work with the object** to allow you to further examine the results.

Parameters

vacation.entities.Employee findById(int)

Parameter	Value
int:	1

Invoke

Results from

- vacation.ejb.interfaces.\_EmployeeFacadeRemote\_Stub.findById()
- vacation.entities.Employee@67ca67ca (vacation.entities.Employee)

Work with Object

- int employeeId
- java.lang.String firstName

Figure 6-24 Results from invoking the EJB in the test client

### Off Course?

If the lookup failed with a message indicating that the schema does not exist (or that the table does not exist within the schema), recheck your JAAS authentication alias setup (4.6.2, “Create the JAAS authentication aliases” on page 103) and your JDBC data source configurations (4.6.4, “Create the Vacation JDBC data source” on page 112).

For example, if you incorrectly specified the DonateAlias for the jdbc/Vacation data source, the access would fail with a message similar to the following:

```
Schema 'DONATE' does not exist {SELECT t0.FIRST_NAME, t0.LAST_NAME,
t0.MIDDLE_NAME, t0.salary, t0.VACATION_HOURS FROM Employee t0 WHERE
t0.EMPLOYEE_ID = ?} [code=-1, state=42Y07]
```

5. Work with the entity JPA instance by expanding on the left to **Objects** → **Employee@67ca67ca (Employee)**
  - a. Select the **String getLastName** getter method from the left hand pane.

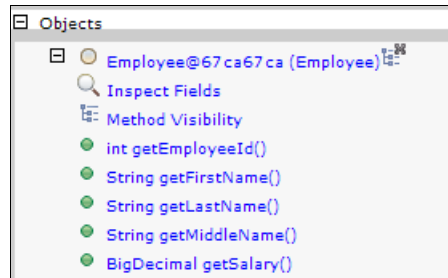


Figure 6-25 Work with the entity JPA instance

- b. From the resulting right Parameters pane, click **Invoke**.
  - View the result (in the bottom half of the right hand pane).

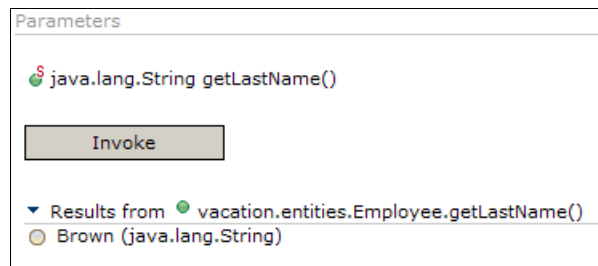


Figure 6-26 Results

- c. Select the **void setVacationHours** setter method from the left hand pane.
  - i. In the right hand pane (Figure 6-27), under **Value**, enter a new value (such as 21).
  - ii. Click **Invoke**. Note successful completion method.

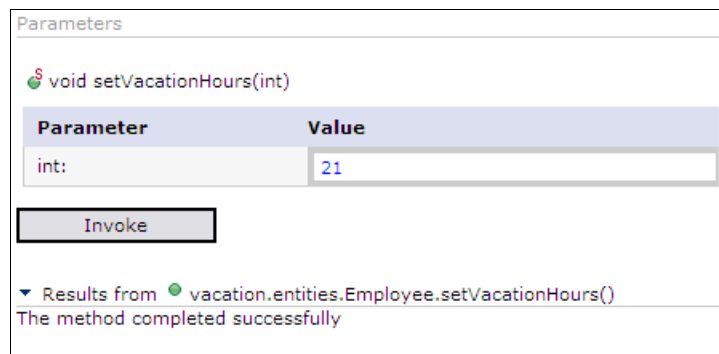


Figure 6-27 Results from the setVacationHours method

- d. Close the UTC Test Client.

## 6.7 Create the EJB 3.0 stub classes

In the new specification for EJB 3.0, the stub classes are no longer needed. In EJB 3.0 the application server itself takes responsibility for keeping track of the Enterprise Java Beans and how to instantiate them when required.

However when creating a thin client (as the JUnit test classes) or an application client, or when you need to interface with a separate application server that does not support EJB 3.0 you need to generate the stub classes. Otherwise, the EJB access will fail with an error similar to the following:

```
java.lang.ClassCastException: Unable to load class:  
vacation.ejb.interfaces._EmployeeFacadeRemote_Stub
```

WebSphere Application Server provides the createEJBStubs command which generates the appropriate stub classes in the EJB jar or EAR file. This script cannot be called directly from with Rational Application Developer to create the stubs within the workspace projects. Instead, you must export the EAR file, run the command, import the updated EJB jar file with the EJB stub classes, and update the JUnit project build path to point to this EJB jar file.

This section describes how you can use the ANT build capability to automate these steps.

More information about this integration between Rational Application Developer, createEJBstubs, and ANT is available at:

<http://www.ibm.com/support/docview.wss?uid=swg21393419>

### 6.7.1 Create the build.xml ANT script

In this section you create the build.xml ANT script that will be used to generate the EJB stubs.

1. In the Package Explorer, select **DonateUnitTester**, right-click and select **New → File**.

- a. At the File pop-up, set File name to **build.xml** and click **Finish**.

The build.xml editor will automatically open.

2. Select anywhere in the white space of the build.xml editor and insert the **F06.5 build.xml** snippet from the Snippets view (Example 6-7).
3. Save and close build.xml.

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

<project basedir="." default="clean">

    <target name="build">
        <earExport earprojectname="DonateEAR"
earExportFile="${basedir}/DonateEAR_zzzz.ear" exportSource="true"
overwrite="true" />
    </target>

    <target name="detect-os">
        <condition property="is.windows" value="true">
            <os family="windows" />
        </condition>
        <condition property="is.unix" value="true">
            <os family="unix" />
        </condition>
        <condition property="suffix" value="bat">
            <isset property="is.windows" />
        </condition>
        <condition property="suffix" value="sh">
            <isset property="is.unix" />
        </condition>
    </target>

    <target name="createstubs" depends="build, detect-os">
        <exec
executable="C:\IBM\SDP\Runtimes\BASE_V7\bin/createEJBStubs.${suffix}">
            <arg line="${basedir}/DonateEAR_zzzz.ear" />
        </exec>
    </target>

    <target name="extractJar" depends="createstubs">
        <unzip dest="${basedir}" src="${basedir}/DonateEAR_zzzz.ear">
            <patternset>
                <include name="DonateEJB.jar" />
            </patternset>
        </unzip>
    </target>

    <target name="clean" depends="extractJar">
        <delete file="${basedir}/DonateEAR_zzzz.ear" />
    </target>

</project>
```

---

If you are using linux replace the executable path from the createstubs target to:

`/opt/IBM/SDP/runtimes/base_v7/bin/createEJBStubs.${suffix}`

#### Behind the scenes:

ANT build files consist of a series of stanzas that define tasks to execute along with pre-requisite actions that need to take place before that task is run:

- ▶ The task is called a target
- ▶ The pre-requisites are called dependencies (depends).

The opening statement in the ANT build file defines the final task to execute. in this the `clean` task (which erases the temporary EAR file)

```
<project basedir="." default="clean">
```

ANT reviews the dependency hierarchy and executes all pre-requisite tasks before finally executing the specified tasks:

<b>build</b>	Export the DonateEAR project into a temporary EAR file.
<b>detectOS</b>	Determine the operating system and set the appropriate command extensions.
<b>createstubs</b>	Execute the createEJBStubs command against the temporary EAR file.
<b>extractJAR</b>	Extract DonateEJB.jar from the updated temporary EAR file.
<b>Clean</b>	Erase the temporary EAR file.

## 6.7.2 Create the build.xml ANT run configuration

In this section you create the customized ANT run configuration.

1. Select **build.xml** (in `DonateUnitTester`), right-click and select **Run As** → **3 Ant Build...**

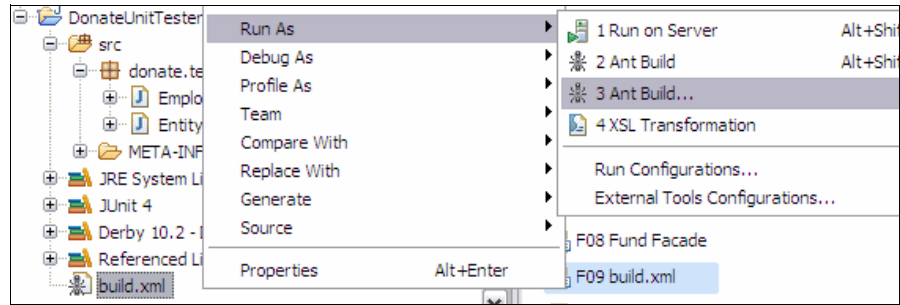


Figure 6-28 Create a new Ant run configuration

**Behind the scenes:** ensure that you select the Ant Build selection that includes the three periods at the end. This will enable you to customize and save the Ant build configuration for use across multiple invocations.

2. At the resulting Edit Configuration pop-up:
  - a. Switch to the **JRE** tab.
  - b. Set the Runtime JRE to **Run in the same JRE as the workspace**.
  - c. At the bottom, click **Apply** and then **Close**.

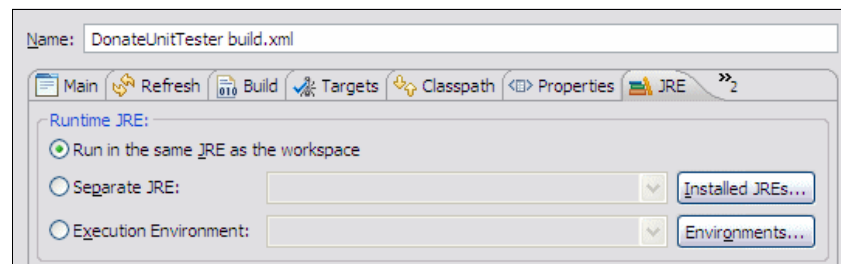


Figure 6-29 Select the JRE parameters

**Behind the scenes:** Several of the ANT tasks used in build.xml (such as earExport) are defined within Rational Application Developer. Therefore, you must run this build.xml in the context of the workbench JRE., otherwise the ANT execution would fail with the following error:

```
BUILD FAILED
C:\workspaces\ExperienceJEE\DonateUnitTester\build.xml:6: Problem:
failed to create task or type earExport
Cause: The name is undefined.
Action: Check the spelling.
Action: Check that any custom tasks/types have been declared.
Action: Check that any <presetdef>/<macrodef> declarations have
taken place.
```

### 6.7.3 Execute the build.xml ANT run configuration

In this section you run the ANT configuration. This section will need to be re-run after you make changes to DonateEJB in order to allow non-container clients to access the updated EJBs.

1. In the Package Explore select **build.xml** (in DonateUnitTester), right-click and select **Run As** → **2 Ant Build**.

**Behind the scenes:** ensure that you select the Ant Build selection that does not include the three periods at the end. This will enable you to run the customized Ant build configuration that you created in 6.7.2, “Create the build.xml ANT run configuration” on page 250.

2. Switch to the console view and verify that the Ant task completed successfully.

```
...
...
extractJar:
    [unzip] Expanding:
C:\workspaces\ExperienceJEE\DonateUnitTester\DonateEAR_zzzz.ear into
C:\workspaces\ExperienceJEE\DonateUnitTester

clean:
    [delete] Deleting:
C:\workspaces\ExperienceJEE\DonateUnitTester\DonateEAR_zzzz.ear
BUILD SUCCESSFUL
Total time: 10 seconds
```



## 6.7.4 Add the EJB stubs to the DonateUnitTester build path

You created DonateEJB.jar in the previous section, and in this section you add it to the Java build path (thus allowing applications to load the classes at execution time).

1. In the Package Explorer select **DonateUnitTester**, right-click and select **Refresh**.
  - Verify that DonateEJB.jar appears in the Package Explorer.



Figure 6-30 DonateEJB.jar in the package

2. In the Package Explorer select **DonateUnitTester**, right-click and select **Properties**.
3. At the resulting Properties pop-up:
  - a. On the left select **Java Build Path** and click **Add Jars**.
  - b. At the resulting JAR selection pop-up, select **DonateUnitTester** → **DonateEJB.jar** and click **OK**.

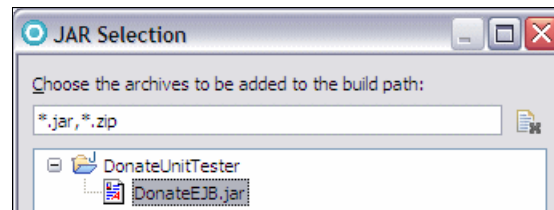


Figure 6-31 Add a JAR file to the build path

- c. Back at the Java Build Path tab, verify that DonateEJB.jar appears in the build path, and click **OK**.

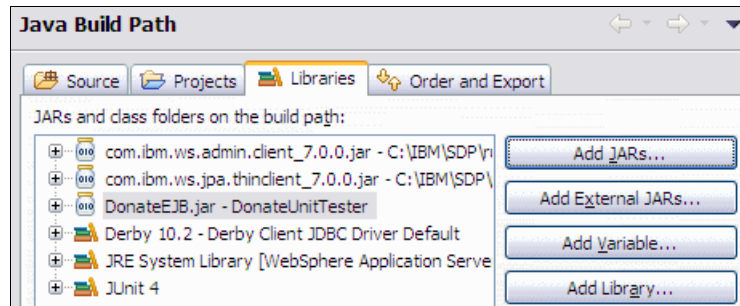
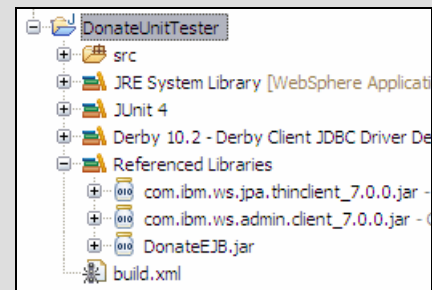


Figure 6-32 Verify the build path

#### Behind the scenes:

DonateEJB.jar will now appear as a referenced library within the Package Explorer. Recall that it previously was displayed as a file resource (appearing below build.xml).



## 6.8 Test the session facades with JUnit

In this section we test the session facade beans with JUnit. We use a similar approach as in 5.8, “Test the Employee and Donate entities” on page 179, and we reuse the same testing project, DonateUnitTester.

### 6.8.1 Create the employee facade tester

To test the employee facade, we create a JUnit test in the DonateUnitTester project. Make sure that you are in the Java perspective:

1. In the Package Explorer, select **donate.test** (in DonateUnitTester/src), right-click and select **New** → **JUnit Test Case**.

**Behind the scenes:** In the Java EE perspective, you have to select **New** → **Other** and then at the pop-up select **Java** → **JUnit** → **JUnit Test Case**

2. At the New JUnit Test Case/JUnit Test Case pop-up:
  - a. Select **New JUnit 4 test**.
  - b. Set the Package to **donate.test**.
  - c. Set the Name to **EmployeeFacadeTester**.
  - d. Click **Finish**.
3. Replace the entire contents of **EmployeeFacadeTester.java** with the **F06.6 EmployeeFacade Tester** snippet from the Snippets view.
4. Save and close EmployeeFacadeTester.java.

#### Behind the scenes:

- The `@BeforeClass` annotated `init` method finds the employee facade through the remote interface (`EmployeeFacadeRemote`):

```
@BeforeClass
@SuppressWarnings("unchecked")
public static void init() throws Exception {
    Hashtable props = new Hashtable();
    props.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");

    ctx = new InitialContext(props);

    java.lang.Object ejbBusIntf = ctx
        .lookup("vacation.ejb.interfaces.EmployeeFacadeRemote");

    employeeFacade = (EmployeeFacadeRemote) javax.rmi.PortableRemoteObject
        .narrow(ejbBusIntf, EmployeeFacadeRemote.class);
}
```

- Two helper methods (`getTestEmployee`) retrieve a test employee with an `employeeId` of 1.

```
public static Employee getTestEmployee
    (EmployeeFacadeInterface employeeFacade) throws Exception {
    return employeeFacade.findEmployeeById(1);
}

public Employee getTestEmployee() throws Exception {
    return getTestEmployee(employeeFacade);
}
```

- The `@Test` annotated `findEmployeeById` method retrieves the employee with `employeeId` 1 through the session facade:

```
public void findEmployeeById() throws Exception {
    Employee e = employeeFacade.findEmployeeById(1);
    Assert.assertNotNull(e);
}
```

- The @Test annotated addHours method donates 100 hours for the test employee, and verifies the result (assertEquals):

```
public void addHours() throws Exception {
    Employee e = getTestEmployee();
    int initialHours = e.getVacationHours();
    e = employeeFacade.addVacationHours(e, 100);
    Assert.assertEquals(e.getVacationHours(), initialHours+100);
}
```

- The @Test annotated deductHours method adds 1000 vacation hours, then removes 500 hours twice. Finally, the method tries to remove more hours than the employee has accumulated, and verifies that the request fails.

```
public void deductHours() throws Exception {
    Employee e = getTestEmployee();
    int initialHours = e.getVacationHours();
    e = employeeFacade.addVacationHours(e, 1000);
    try {
        e = employeeFacade.deductVacationHours(e, 500);
        e = employeeFacade.deductVacationHours(e, 500);
    } catch (Exception ex) {
        Assert.fail("Unexpected exception returned: " + ex.getMessage());
    }
    try {
        e = employeeFacade.deductVacationHours(e, 1+initialHours);
        Assert.fail("Deduction fails for negative hours");
    } catch (Exception ex) {
        System.out.println("deductHours: " + ex.getMessage());
    }
}
```

## 6.8.2 Run the employee facade tester

To run the test, follow the same basic steps used in 5.8.4, “Run the JUnit test” on page 186:

1. In the Package Explorer, select **EmployeeFacadeTester.java** (in DonateUnitTester/src/donate.test), right-click and select **Run As → JUnit Test**.
2. Verify that all tests were successful in the JUnit view. Successful tests are marked by a green check mark.

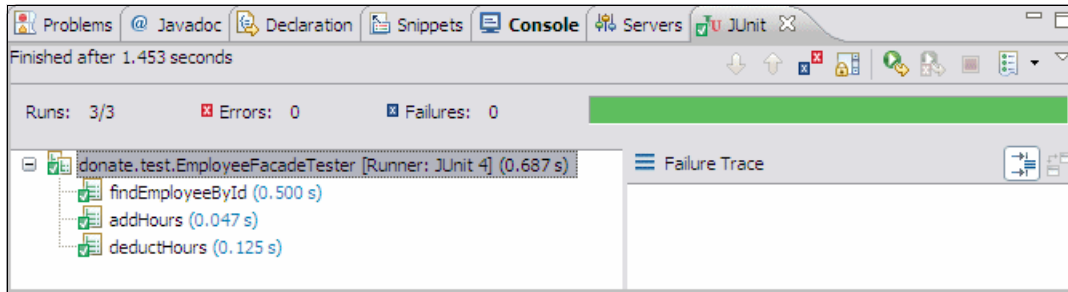


Figure 6-33 Test results in the JUnit view

### Off course:

If you see the following failure:

```
java.lang.NoClassDefFoundError: com.ibm.ffdc.Manager
```

you neglected to add the `com.ibm.ws.admin.client_7.0.0.jar` file to the Java Build Path in 5.8.1, "Create the JUnit project" on page 180. This jar file contains the necessary files for accessing the WebSphere Application Server initial context factory from a non-Java EE client.

### Off course:

If you get the following failure:

```
javax.naming.NameNotFoundException: Context:
romurtaNode02Cell/nodes/romurtaNode02/servers/server1, name:
EmployeeFacadeRemote: First component in name EmployeeFacadeRemote
not found. [Root exception is
org.omg.CosNaming.NamingContextPackage.NotFound:
IDL:omg.org/CosNaming/NamingContext/NotFound:1.0]
```

it is possible the server IIOP port is the default one. To set a different port to be used when connecting to the server use the `PROVIDER_URL` property in the tester code as below:

```
props.put(Context.PROVIDER_URL,"iiop://localhost:PORTNUMBER/");
```

Replacing `PORTNUMBER` by the port used by the IIOP service.

### 6.8.3 Create the fund facade tester

To test the fund facade we create a **FundFacadeTester** JUnit test case in the DonateUnitTester project:

1. In the Package Explorer, select **donate.test** (in DonateUnitTester/src), right-click and select **New** → **JUnit Test Case**.
2. At the New JUnit Test Case/JUnit Test Case pop-up:
  - a. Select **New JUnit 4 test**.
  - b. Set the Package to **donate.test**.
  - c. Set the Name to **FundFacadeTester**.
  - d. Click **Finish**.
3. Replace the entire contents of **FundFacadeTester.java** with the **F06.7 FundFacade Tester** snippet from the Snippets view.
4. Save and close FundacadeTester.java.

### 6.8.4 Run the fund facade tester

To run the test follow the same steps as for the employee facade.

1. In the Package Explorer, select **FundFacadeTester.java** (in DonateUnitTester/src/donate.test), right-click and select **Run As** → **JUnit Test**.
2. Verify that all tests were successful in the JUnit view. Successful tests are marked by a green check mark.

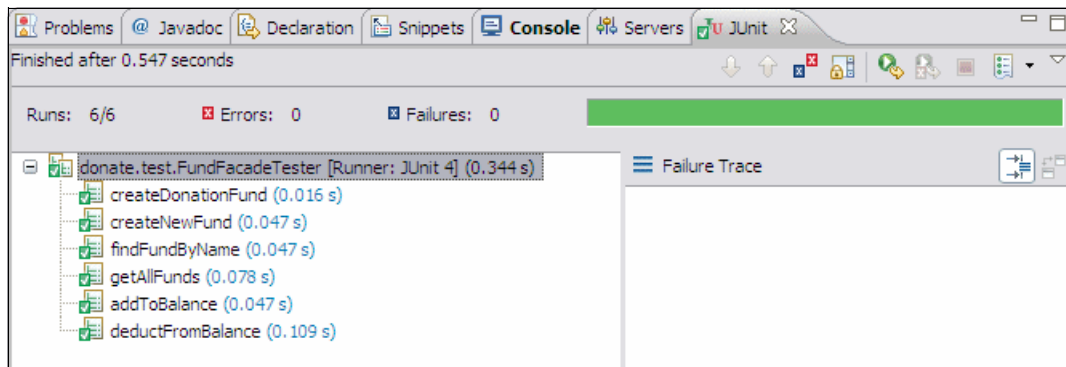


Figure 6-34 Test results in the JUnit view

**Behind the scenes:** Note that when you rerun the FundFacadeTester, the createDonationFund test fails because the DonationFund already exists in the database:

```
java.lang.Exception: Fund name already exists
    at donate.ejb.impl.FundFacade.createNewFund(FundFacade.java:48)
    .....

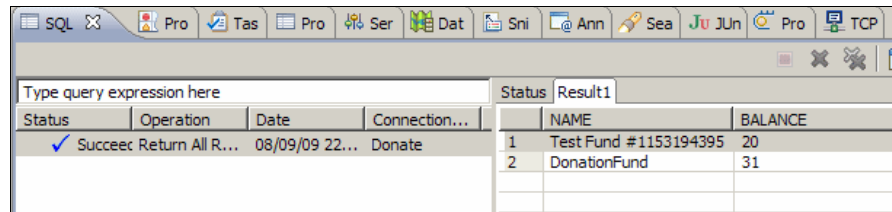
```

The other tests always succeed.

## 6.9 Verify the databases

After running the JUnit test cases for the entity facade session beans, the FUND table is created in the Donate database, and it contains the DonationFund and some random mock funds.

1. From the Java EE perspective Data Source Explorer view select **Databases** → **Donate (Apache Derby xxx)** → **Donate** → **Schemas** → **DONATE** → **Tables** → **FUND**, right-click and select **Data** → **Return All Rows**
2. Switch to the SQL Results view (you might have to resize it), and then select the **Result1** tab on the left. The DonationFund should now be defined.



Status	Operation	Date	Connection...
✓ Success	Return All R...	08/09/09 22...	Donate

	NAME	BALANCE
1	Test Fund #1153194395	20
2	DonationFund	31

Figure 6-35 Database verification results

## 6.10 Explore!

Additional Explore resources about EJB 3.0 session beans are available in the IBM Redbooks publication, *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611.







## Create the Donate session bean for the business logic

In this chapter, we create a new session EJB that contains the business logic of the application. This new session EJB transfers vacation hours from an employee record to a funds record.

## 7.1 Learn!

Figure 7-1 shows the new session EJB which transfers vacation hours from an employee record to a funds record.

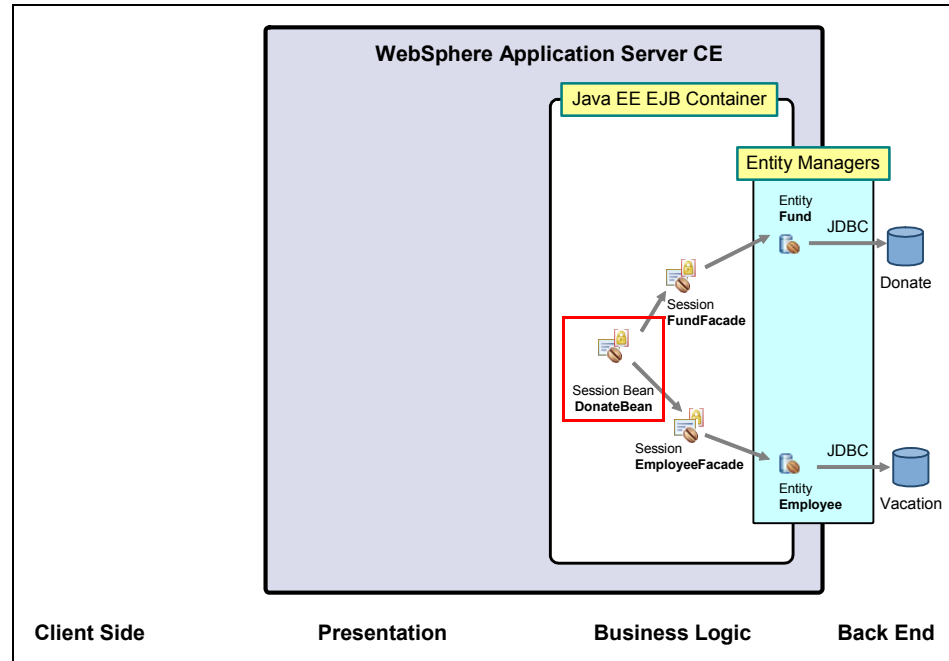


Figure 7-1 Donate application: Session bean for business logic

The Learn resources for this chapter are described in Chapter 6, “Create the entity facade session beans” on page 201.

## 7.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed these steps:

- ▶ Defined the two databases (Vacation and Donate)
- ▶ Defined the matching database pools (VacationPool and DonatePool)
- ▶ Created the JPA entities (Employee and Fund)
- ▶ Created the two session facade beans (EmployeeFacade and FundFacade)

- Created and run the JUnit test cases for JPA entities and the facades

## 7.3 Define the Donate session EJB

In this section, we define the Donate session EJB in the existing DonateEJB project. We follow the same approach as in 6.4, “Create the entity facade interfaces and session beans” on page 225.

### 7.3.1 Define the business interface for the Donate session bean

We define the business interface that is implemented by the Donate session bean in the same way as for the employee facade (see 6.4.1, “Define the business interface for the employee facade” on page 225):

1. In the Java EE perspective Enterprise Explorer select **DonateEJB**, right-click and select **New** → **Interface**.
2. At the New Java Interface/Java Interface pop-up:
  - a. Set Package to **donate.ejb.interfaces**.
  - b. Set name to **DonateBeanInterface**.
  - c. Click **Finish**.
3. In the resulting DonateBeanInterface.java editor, move the cursor to the interface body (before the closing brace in the file) and insert the following method definition:

```
public interface DonateBeanInterface {  
    public void donateToFund(Employee employee, Fund fund, int hours)  
                                   throws Exception;  
}
```

4. Organize imports (**Ctrl+Shift+O**) to resolve:

```
import vacation.entities.Employee;  
import donate.entities.Fund;
```
5. Save and close DonateBeanInterface.java.

### 7.3.2 Define the remote interface for the Donate session bean

For access by remote clients, we also define a remote interface for the session bean (see 6.4.2, “Define the remote interface for the employee facade” on page 227):

1. In the Enterprise Explorer, select **donate.ejb.interfaces** (in DonateEJB/ejbModule), right-click and select **New** → **Interface**.
2. At the New Java Interface pop-up,

- a. Set Name to **DonateBeanRemote**.
  - b. Next to Extended Interfaces, and click **Add**.
    - At the Extended Interfaces Selection pop-up, begin typing **DonateBeanInterface** until it appears in the Matching item list. Select it from Matching items and click **OK**.
  - c. Back at the New Java Interfaces pop-up, note that DonateBeanInterface is now listed in the Extended interfaces list.
  - d. Click **Finish**.
3. In the resulting DonateBeanRemote.java editor, add a **@Remote** annotation above the method signature:

```
@Remote
public interface DonateBeanRemote extends DonateBeanInterface {
}
```

4. Move the cursor to the interface body (before the closing brace) and insert the following method definition:

```
public interface DonateBeanRemote extends DonateBeanInterface {
    public String donateToFund(int employeeId, String fundName,
                           int hours);
}
```

#### Behind the scenes:

Note that the method unique to the remote interface uses simple argument types:

```
public String donateToFund(int employeeId, String fundName, int hours)
```

Where the method in the base interface (valid for both local and remote access) takes complex types:

```
public void donateToFund(Employee employee, Fund fund, int hours)
```

Remote callers (running outside of the Java EE container) such as Web clients, Web services, or message-driven beans are less likely to have access to the full object definition. Therefore, it is preferable to allow them to send the least amount of information required to perform the operation.

5. Organize imports (**Ctrl+Shift+O**) to resolve `javax.ejb.Remote`.
6. Save and close DonateBeanRemote.java.

### 7.3.3 Create the Donate session bean

In this section we create the Donate session bean in the same way as we created the session facades (see 6.4.3, “Create the session bean facade for the employee entity” on page 228):

1. In the Enterprise Explorer, select **DonateEJB**, right-click and select **New** → **Session Bean**.
2. At the Create EJB 3.0 Session Bean pop-up:
  - a. Set Java package to **donate.ejb.impl**.
  - b. Set Class name to **DonateBean**.
  - c. Clear the **Create business interface** check boxes.
  - d. Click **Next**.
3. At the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, click **Add**.
  - a. At the Choose Interface pop-up, begin typing **DonateBeanInterface** until it appears in the Matching item list. Select it from Matching items
  - b. Under Add as select **Local**.
  - c. Click **OK**.
4. Back at the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, click **Add**.
  - a. At the Choose Interface pop-up, begin typing **DonateBeanRemote** until it appears in the Matching item list. Select it from Matching items
  - b. Under Add as select **Remote**.
  - c. Click **OK**
5. Back at the Create EJB 3.0 Session Bean/Enter Session Bean specific information pop-up, note that both the local and remote interface appears, and click **Next**.
6. At the Create EJB 3.0 Session Bean/Select Class Diagram for Visualization:
  - a. Clear **Add bean to Class Diagram**.
  - b. Click **Finish**.
7. The DonateBean.java editor opens automatically with the generated skeleton session bean (Example 7-1 on page 265).

*Example 7-1 Skeleton donate session bean*

---

```
package donate.ejb.impl;
import vacation.entities.Employee;
import donate.ejb.interfaces.DonateBeanInterface;
import donate.ejb.interfaces.DonateBeanRemote;
import donate.entities.Fund;
```

```

@Stateless
@Local( { DonateBeanInterface.class })
@Remote( { DonateBeanRemote.class })
public class DonateBean implements DonateBeanInterface, DonateBeanRemote {

    public void donateToFund(Employee employee, Fund fund, int hours)
        throws Exception {
        // TODO Auto-generated method stub

    }

    public String donateToFund(int employeeId, String fundName, int hours) {
        // TODO Auto-generated method stub
        return null;
    }
}

```

---

### Behind the scenes:

Note that two different donateToFund method skeletons are created, one from the base interface and one from the remote interface.

8. Delete the generated skeleton methods. Note that the generated comments lines were omitted from the deleted fragment shown below.

```

public String donateToFund(int employeeId, String fundName, int hours) {
    // TODO Auto-generated method stub
    return null;
}

/ public void donateToFund(Employee employee, Fund fund, int hours) {
    // TODO Auto-generated method stub
}

```

9. Before the closing brace, insert the **F07.1 DonateBean** snippet from the Snippets view. Refer to the following **Behind the scenes** box for details about the inserted methods.
10. Organize imports (**Ctrl+Shift+O**) to resolve:

```

import javax.ejb.EJB;
import javax.ejb.Local;
import javax.ejb.Stateless;

```

11. Save and close DonateBean.java.

This concludes the creation of the DonateBean session bean.

### Behind the scenes:

- The @EJB annotated interfaces define references to the business interfaces of the two facade session bean, EmployeeFacade and FundFacade, which are used to access the databases:

```
@EJB
private EmployeeFacadeInterface employeeFacade;

@EJB
private FundFacadeInterface fundFacade;
```

The @EJB annotation uses EJB 3.0 injection to define the references to the other session beans, without using deployment descriptors.

- The first donateToFund method deducts the donated hours from the employee and adds the hours to the fund. The parameters are the Employee and FUND JPA entity objects. Exceptions are returned to the caller.



```
public void donateToFund(Employee employee, Fund fund, int hours)
    throws Exception {
    employeeFacade.deductVacationHours(employee, hours);
    fundFacade.addToBalance(fund, hours);
}
```

- The second donateToFund method performs the same function, but first has to retrieve the employee and fund objects using the given keys. This method handles any exceptions and returns an error message.

```
public String donateToFund(int employeeId, String fundName,
                           int hours) {
    String return_string = null;
    try {
        Employee employee = employeeFacade
                               .findEmployeeById(employeeId);
        Fund fund = fundFacade.findFundByName(fundName);
        donateToFund(employee, fund, hours);
        return_string = "Transfer successful";
    } catch (Exception e) {
        e.printStackTrace();
        return_string = "Error: " + e.getMessage();
    }
    return return_string;
}
```

## 7.4 Test the Donate session EJB with the UTC

In this section you use the UTC to test the Donate session EJB using the remote interface.

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to
    -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`
  - and execute:
    -  `startNetworkServer.bat`
    -  `startNetworkServer.sh`
  - If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
    - If the server instance starts successfully, you will find the message `WSVR0001I: Server server1 open for e-business` in the Console view.

### Behind the scenes:

Recall that you started both of these in earlier chapters. They should still be running unless you stopped them or restarted the system.

- ▶ 4.2, “Start the Derby Network Server” on page 87.
- ▶ 3.6, “Start the ExperienceJEE Test Server” on page 78.

2. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
3. Restart the **UTC** by selecting the server, right-clicking and selecting **Universal test client** → **restart**.
  - a. Log in using the **javaeeadmin** User ID and password that you specified in section 3.4, “Create the ExperienceJEE server profile” on page 62.
4. In the resulting **Test Client**, in the upper left select **JNDI Explorer**.
5. In the **JNDI Explorer** pane (in the right) open the reference to the entity EJB remote home interface by selecting **ejb** → **DonateEAR** → **DonateEJB.jar** → **DonateBean#donate.ejb.interfaces.DonateBeanRemote** (**donate.ejb.interfaces.DonateBeanRemote**).



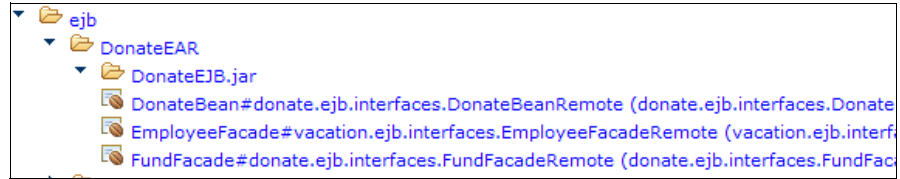


Figure 7-2 Open the entity EJB remote home interface

6. In the **EJB Beans** section (on the left), select **EJB Beans** → **DonateBeanRemote** → **DonateBeanRemote** → **String donateToFund**

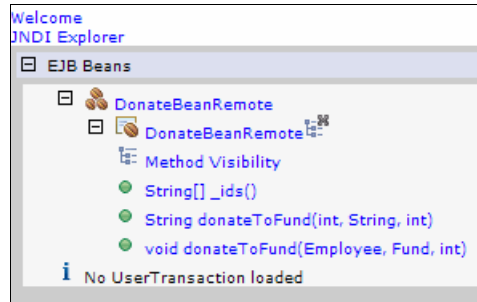




Figure 7-3 Open the method

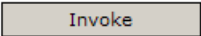
7. In the resulting right panel:
  - a. Under the **Value** column in the first row (for the employee ID) enter **1**.
  - b. Under the **Value** column in the second row (for the fund name) enter **DonationFund**.
  - c. Under the **Value** column in the third row (for the transfer amount) enter **1**.
  - d. Click **Invoke**.

The resulting message should be “Transfer Successful”. This text came from the donateVacation method that you created in DonateBean.java.

Parameters

 java.lang.String donateToFund(int, String, int)

Parameter	Value
int:	1
String:	DonationFund 
int:	1



▼ Results from



-  donate.ejb.interfaces.\_DonateBeanRemote\_Stub.donateToFund()
-  Transfer successful (java.lang.String)

Figure 7-4 Enter the test values

### Off course?

If the retrieve fails expand examine both the failure message within the UTC and any failure messages within the Console view.

The most common error is an incorrect JDBC configuration, and in particular accessing a one-phase JDBC provider. This results in the following UTC message:

```
javax.ejb.EJBTransactionRolledbackException:
```

and the associated Console view message:

```
[12/28/09 13:26:07:888 EST] 00000044 RegisteredRes E WTRN0063E: An  
illegal attempt to commit a one phase capable resource with existing  
two phase capable resources has occurred.
```

This occurs when you instruct Rational Application Developer to configure the project for JDBC deployment, and as a result it configures a one phase JDBC data source/provider. This deploy time configuration overrides the two-phase configuration you made in the ExperienceJEE Test Server in 4.6, “Configure the ExperienceJEE Test Server” on page 100.

Resolution:

1. In the Java EE perspective Enterprise Explorer, delete **ibmconfig** (in DonateEAR/META-INF).
2. Clean the project deployment (In the servers, view, select **ExperienceJEE Test Server**, right-click and select **Clean**)
3. Restart the **ExperienceJEE Test Server**.

4. Repeat the **donateVacation** invocation but this time use an employee ID that does not exist (such as 8). Note that the exception (of not finding the Employee record) is handled by the code in your session EJB, and a message is returned to the requester.

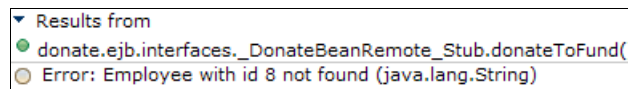


Figure 7-5 Exception results when an ID doesn't exist

5. Repeat the **donateVacation** invocation but this time use an employee ID that DOES exist (such as 1) and a fund name that DOES NOT exist (such as AnotherFund). The exception of not finding the Fund record is handled by the code in your session EJB, and a message is returned to the requester.

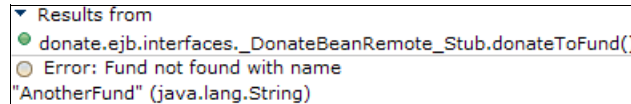


Figure 7-6 Exception results when a fund name doesn't exist

6. Close the UTC.

#### Extra credit:

You could also use UTC and FundsFacade to verify that the Funds entity EJB was updated.

## 7.5 Test the session bean with JUnit

In this section we test the session facade beans with JUnit. We use a similar approach as in 6.8, “Test the session facades with JUnit” on page 254, and we reuse the same testing project, `DonateUnitTester`.

### 7.5.1 Update the EJB stubs

The JAR file that contains the EJB stubs (that you created in 6.7, “Create the EJB 3.0 stub classes” on page 248) must be updated because of the new `Donate` session EJB.

1. In the Java perspective Package Explore select **build.xml** (in `DonateUnitTester`), right-click and select **New** → **2 Ant Build**.
2. Switch to the console view and verify that the Ant task completed successfully.
3. In the Java perspective Package Explore select **DonateUnitTester**, right-click and select **Refresh**.

### 7.5.2 Create the Donate bean test case

To test the `DonateBean` session bean, we create a JUnit test in the `DonateUnitTester` project. Make sure that you are in the Java perspective:

1. Select **donate.test** (in `DonateUnitTester/src`), right-click and select **New** → **JUnit Test Case**. Note that in the Java EE perspective, you have to select **New** → **Other** and at the pop-up, select **Java** → **JUnit** → **JUnit Test Case**.
2. At the New JUnit Test Case/JUnit Test Case pop-up:

- a. Select **New JUnit test**.
  - b. Set the Package to **donate.test**.
  - c. Set the Name to **DonateBeanTester**.
  - d. Click **Finish**.
3. Replace the entire contents of **DonateBeanTester.java** with the **F07.2 DonateBean Tester** snippet from the Snippets view.
4. Save and close `DonateBeanTester.java`.

#### Behind the scenes:

- The `@BeforeClass` annotated `init` method finds the session bean and the entity facades through the remote interfaces (`DonateBeanRemote`, `FundFacadeRemote`, and `EmployeeFacadeRemote`):

```
public static void init() throws Exception {
    System.out.println("DonateBeanTester: Start");
    Hashtable props = new Hashtable();

    props.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.websphere.naming.WsnInitialContextFactory");
    ctx = new InitialContext(props);
    donateBean =
        (DonateBeanRemote)ctx.lookup("donate.ejb.interfaces.DonateBeanRemote");
    fundFacade =
        (FundFacadeRemote)ctx.lookup("donate.ejb.interfaces.FundFacadeRemote");
    employeeFacade =
        (EmployeeFacadeRemote)ctx.lookup("vacation.ejb.interfaces.EmployeeFacadeRemote");
}
```

- The `@Test` annotated `donateToFund` method creates a mock fund, retrieves an employee through the session facade, adds 1000 hours to the employee, donates 1000 hours to the mock fund, and verifies that the employee and fund have been updated:

```
public void donateToFund() {
    try {
        Fund fund = FundFacadeTester.createMockFund(fundFacade, cleanup);
        Employee employee = EmployeeFacadeTester
            .getTestEmployee(employeeFacade);
        employee = employeeFacade.addVacationHours(employee, 1000);
        donateBean.donateToFund(1, fund.getName(), 1000);
        assertEquals(
            fundFacade.findFundByName(fund.getName()).getBalance(),
            fund.getBalance()+1000
        );
        assertEquals(
            employeeFacade.findEmployeeById
                (employee.getEmployeeId()).getVacationHours(),
            employee.getVacationHours()-1000
        );
    } catch (Exception e) {
        fail("Unexpected exception thrown: " + e.getMessage());
    }
}
```

### 7.5.3 Run the Donate bean tester

To run the test, follow the steps outlined in 5.8.4, “Run the JUnit test” on page 186:

1. In the Servers view of the Java EE perspective, publish the DonateEAR application (select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. In the Project Explorer, select **DonateBeanTester.java** (in `DonateUnitTester/src/donate.test`), right-click and select **Run As → JUnit Test**.
3. Verify that all tests were successful in the JUnit view. Note that every test run increases the balance in the DonationFund by one hour. You can verify that in the workbench Console using the DataSource Explorer.

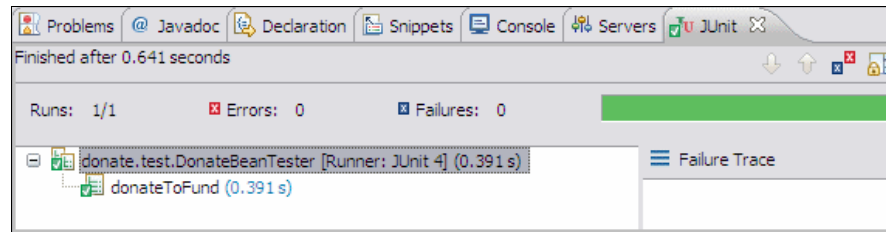


Figure 7-7 Test results in the JUnit view

## 7.6 Explore!

I

There are no unique Explore resources for this chapter.







## Create the Web front end

In this chapter, we present two Web applications, Basic and JSF (Figure 8-1 on page 278).

In 8.3, “Basic Web application” on page 286, you gain hands-on experience in creating servlets and JSPs, which will interact with the `EmployeeFacade` and `DonateBean` session EJBs created in the previous chapters. The Basic Web application section is not required for any other chapters in this book, but completing this section illustrates how a Web application connects to EJB 3.0 session beans through simple injection.

In 8.4, “JSF Web application” on page 297, you gain hands-on experience in creating a JSF application, which will interact with the `EmployeeFacade` and `DonateBean` session EJBs created in the previous chapters.

## 8.1 Learn!

Figure 8-1 shows the Donate applications Web client front-end.

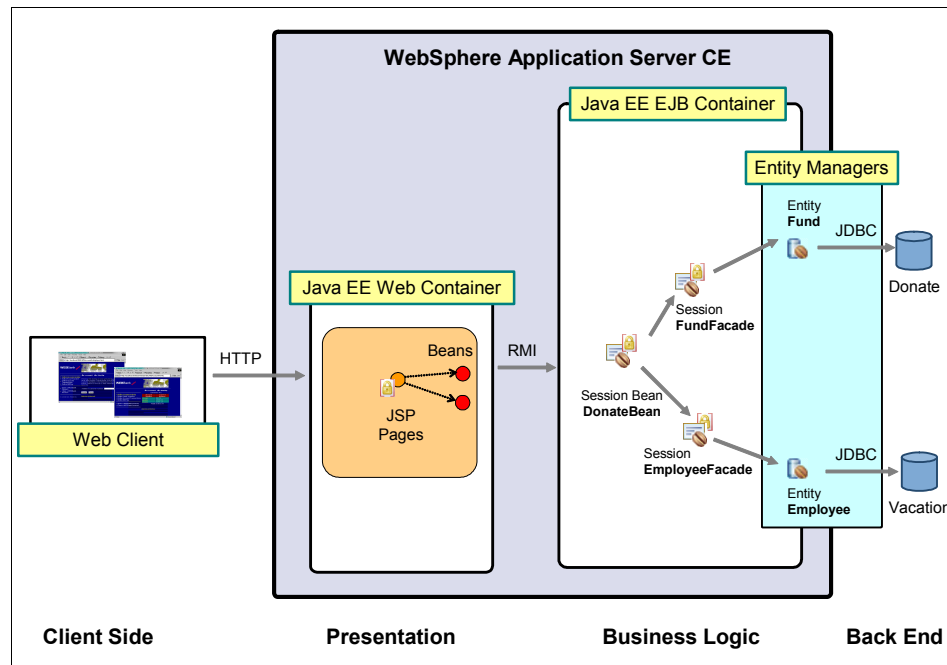


Figure 8-1 Donate application: Web front end

In general, an overall *Web front end* in Java EE usually consists of **Web components**, which are the actual artifacts that are invoked when a user types a Web address into a browser. In a Java EE environment, the addressable Web components can include these components:

- ▶ Static content, such as HTML pages or image files. Note that these are not included in the Java EE specifications, but they certainly provide a core portion of an overall Web front end, and they can be utilized side-by-side with the actual Java EE artifacts.
- ▶ Servlets, which provide 100% *dynamic* content.
- ▶ JavaServer Pages (JSPs), which provide for *partial dynamic* content. The JSP consists of both HTML statements and Java code.
- ▶ Session store, which maintains persistence information between the invocation of various Web components.
- ▶ Controller, which acts as an intermediary between the request and the actual final page.

## 8.1.1 What is a servlet?

A servlet is a Java EE specification which is used to extend the capabilities of servers that host applications accessed via a request-response programming model. In theory, servlets can respond to any type of request. The most common usage of servlets is to extend the applications hosted by Web servers. In this case, the underlying protocol with which the servlet interacts is HTTP.

A servlet receives a request, returns a response, or forwards processing to another servlet or a JSP for the response. In the request, data is passed to the servlet through parameters and attributes. Parameters are strings, whereas attributes can be any objects. The response is typically written in HTML format for rendering in a browser. A conversation can be maintained with a user by saving data (objects) as session data that is available in subsequent requests.

### Servlet life cycle

In order to fully understand servlets, it is important to understand their life cycle. Like all Java EE container-based technologies, a servlet's life cycle is controlled by a container (in this case, a Web container). These are the life cycle steps of a servlet:

- **Start up:** If an instance of the servlet class does not exist, the Web container creates an instance of the servlet class. As a part of this step, the Web container also invokes the servlet's `init` method.
- **Handling the request:** An in-bound request (regardless of whether a GET or a POST) is handled by the servlet's `service` method. Typically the service method forwards the request to either the `doGet` or `doPost` method. A sample `doGet` method is shown here:

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("<body>");
    out.println("<h1>Hello World</h1>");
    out.println("</body>");
}
```

- **Shut down:** When the servlet instance is ready to be removed (or recycled), the container invokes the `destroy` method of the servlet.

For more in-depth discussion on the servlet specification, refer to JSR 53: Servlet 2.3 and JSP 1.2 specification at:

<http://jcp.org/en/jsr/detail?id=53>

### 8.1.2 What is a JSP?

A JavaServer Page (JSP) is a text-based document that contains two types of text:

- ▶ Static template data, which can be expressed in any text-based format, such as HTML, SVG, WML, and XML
- ▶ JSP elements, which construct dynamic content. At runtime, JSPs are compiled into servlets.

### 8.1.3 Session store

The session store maintains information between the client (a browser) and the server:

- ▶ On the client side, the session ID is typically identified through a cookie (JSESSIONID) that is stored in the browser and transmitted in the HTTP header.
- ▶ On the server side, the session information is maintained in a persistence store (typically in memory, but also potentially in a relational database or in a publish/subscribe mechanism).

The first artifact invocation initializes the store and returns the session ID to the browser. On subsequent artifact invocations, this session ID is used to retrieve any stored information from the session store.

### 8.1.4 Controller

In basic Web development, there are initially two general Web development paradigms:

- ▶ **Model 1:** The Model 1 approach, the simpler of the two paradigms, is based on the notion that a request is made to a JSP or servlet. The JSP or servlet then handles all responsibilities for the request, including processing the request, validating data, handling the business logic, and generating a response.
- ▶ **Model 2:** The Model 2 approach employs the concept of separation of concerns. Specifically, requests from the client browser are passed to the controller, which is a servlet. The controller then decides which view (JSP) to pass the request to create the response.

In general, small applications can be written using the Model 1 approach. Larger enterprise-ready applications are better served using a Model 2 approach. The Model 2 approach is called *model-view-controller* (MVC).

## 8.1.5 Model-view-controller

Model-view-controller (MVC) is an architectural pattern. The pattern is grounded on the concept of isolating business logic layer from user interface layer (through a controller layer). The main benefit of employing such an approach is the application is not monolithic (and therefore easier to modify and maintain). It also enforces delegation of responsibilities between team members in a development team:

- ▶ **Model:** The model represents the information (the data) of the application and the business rules used to manipulate the data. The model is implemented as Java components, such as EJBs and JavaBeans, where EJBs provide the business logic and JavaBeans are the data transfer objects (DTO). The model encapsulates application state, responds to state queries, exposes application functionality, and notifies the view of changes.
- ▶ **View:** The view corresponds to elements of the user interface such as text and images, and it typically implemented as JSPs. The view renders the model, requests updates from the model, sends user input to the controller, and allows the controller to select the view.
- ▶ **Controller:** The controller handles all the requests, manages the communication to the model, and passes control to the view JSPs for the response to the client. The controller defined application behavior, maps user actions to model updates, and selects the view for the response.

Figure 8-2 shows a typical implementation of the model-view-controller architecture using servlets, JSPs, and EJBs.

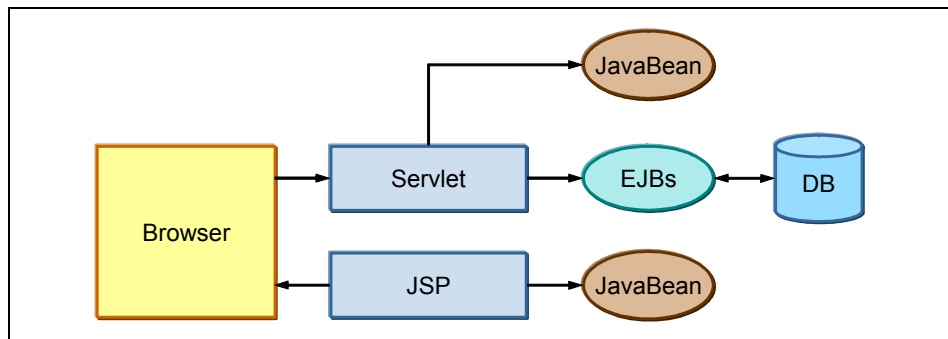


Figure 8-2 Model-view-controller architecture using servlets and JSPs

Note that out-of-the-box, the MVC programming paradigm is not enforced in servlet/JSP programming. MVC is an architectural approach: Java EE provides the required technical components, but the developer still must manually create quite a bit of code to coordinate the overall application.

The open source community developed a framework called Struts that defined the artifacts and required runtime support needed for a flexible MVC oriented Web application. This framework has been very popular and is shipped with many Java EE runtimes and Java EE IDEs. Note that Struts is not a standard or specification: It is simply an implementation that has been widely adopted. Other MVC frameworks are Spring MVC and Tapestry.

Java EE includes the JavaServer Faces (JSF) specification, which provides similar support.

### 8.1.6 What is a JSF?

In this section we describe the JavaServer Faces technology and develop a JSF-based Web application. JSF is part of the J2EE and Java EE specification and addresses many shortcomings of previous Web technologies.

JavaServer Faces technology, which is built on top of servlet/JSP technology, was created to simplify Web development, while enforcing best practices (such as promoting re-use and separation of concerns). Furthermore, JSF provides a component-based platform for Web development.

### 8.1.7 JSF application architecture

The JSF application architecture can be easily extended in a variety of ways to suit the requirements of your particular application. You can develop custom components, renderers, validators, and other JSF objects and register them with the JSF runtime.

The focus of this section is to highlight the JSF application architecture shown in Figure 8-3 on page 283:

- ▶ **Faces JSP pages:** Pages are built from JSF components, where each component is represented by a server-side class.
- ▶ **Faces servlet:** One servlet (`FacesServlet`) controls the execution flow.
- ▶ **Configuration file:** An XML file (`faces-config.xml`) that contains the navigation rules between the JSPs, validators, and managed beans.
- ▶ **Tag libraries:** The JSF components are implemented in tag libraries.
- ▶ **Validators:** Java classes to validate the content of JSF components, for example, to validate user input.
- ▶ **Managed beans:** JavaBeans defined in the configuration file to hold the data from JSF components. Managed beans represent the data model and are passed between business logic and user interface. JSF moves the data between managed beans and user interface components.

- **Events:** Java code executed in the server for events (for example, a push button). Event handling is used to pass managed beans to business logic.

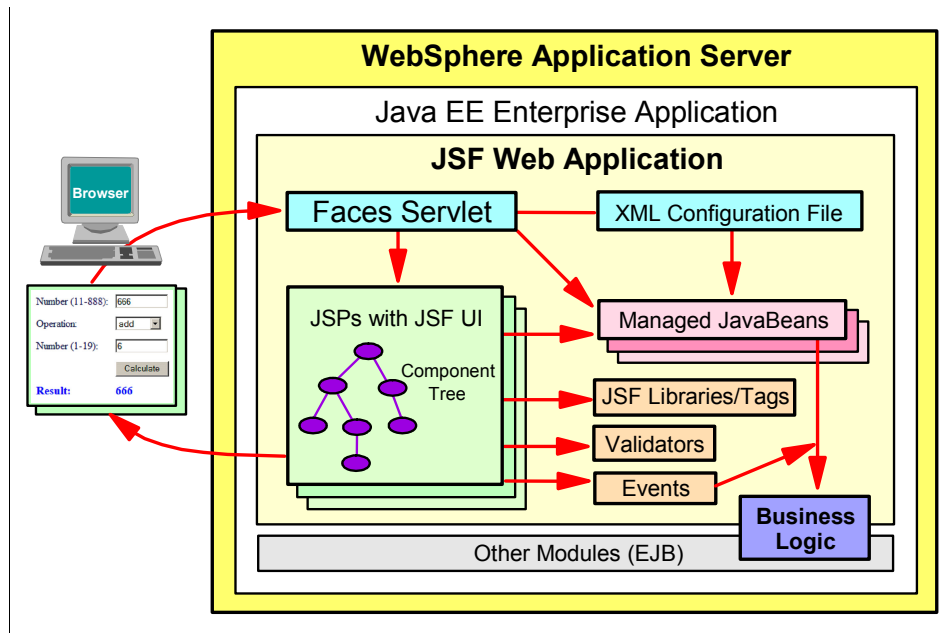


Figure 8-3 JSF application architecture

## 8.1.8 JSF life cycle

In an attempt to reduce complexity around servlets and provide a specification-backed standard framework, JSF was created to plug holes in servlets. Based on servlets, JSF embodies a more complex life cycle (Figure 8-4) processing that addresses typical processing in Web applications.

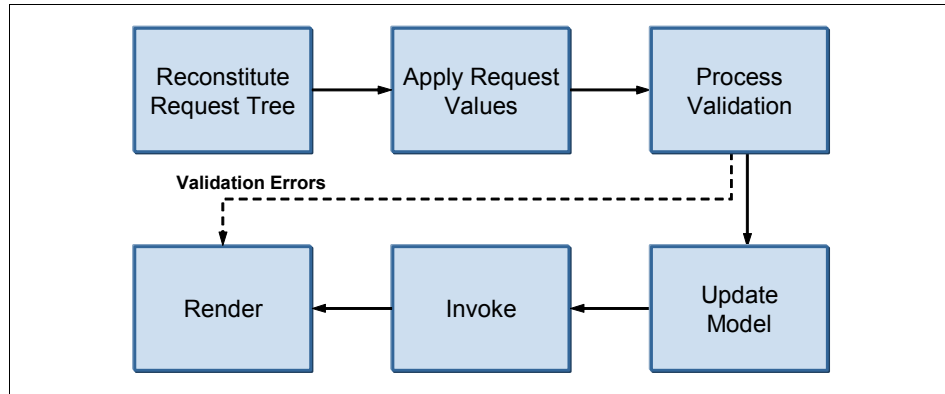


Figure 8-4 JSF lifecycle

- ▶ **Start up** (not shown): If an instance of the JSF servlet class does not exist, the Web container creates an instance of the JSF servlet class. As a part of this step, the Web container invokes the JSF servlet's `init` method.
- ▶ **Reconstitute Request Tree**: In this phase, an inbound request is handled by the Faces servlet. The controller extracts the view ID (from request), which is determined by the name of the JSP page. The Faces servlet uses the view ID to identify the component(s) for the current view. This phase of the life cycle presents three view instances: New, initial, and postback, with each being handled differently. New and initial view are related to creation and the initial loading the view. The postback view is primarily for restoring views in cases of failed user interface validations.
- ▶ **Apply Request Values**: In this phase, each component discovers its the current state by retrieving current values from either request parameters, cookies or headers. This phase also supports an immediate event handling property. This property of JSF is to handle events that do not require form validations.
- ▶ **Process Validation**: In this phase, each component will have its values validated against the application's validation rules. Should a validation fail, an error message is added to the Faces context, and the component is marked invalid. Thereafter, JSF advances to the render response phase, which will display the current view with the validation error messages. If there are no validation errors, JSF advances to the next phase.



- ▶ **Update Model:** In this phase, the values collected are applied to the managed beans. Only bean properties that are bound to a component's value will be updated.
- ▶ **Invoke:** In this phase, we can invoke the business logic. In addition, we can also specify the next logical view (or navigation) for a given sequence or number of possible sequences.
- ▶ **Render:** In the last phase, the results of the model are displayed.
- ▶ **Shut down** (not shown): When the JSF servlet instance is ready to be removed (or recycled), the container invokes the destroy method of the JSF servlet.

For more detailed accounting on JSF, refer to the JSR 127 specification at:

<http://jcp.org/en/jsr/detail?id=127>

### 8.1.9 Goals of JSF

The obvious next question now is, if JSF is built on top of servlets/JSPs, then what is the value-add of JSF (why do we need it)?

To better understand the motivation behind JSF, consider some of the gaps that exist in the Web application development space using other Java EE technologies.

In no particular order, here are some gaps in available technologies:

- ▶ There are no standards around building custom widgets for building a complex Web based user interface. For example, consider a graphical user interface that uses a tree hierarchy view. UI components would need to be designed and built to support this type of interface. In addition, the design of these components should be extensible and reusable in order to accommodate any future functionality and maintenance of the product.
- ▶ The same lack of standards in building custom widgets also makes it difficult for Tool providers to build IDEs that provide developers with an easier development experience and make Java EE technologies more accessible to a wide variety of development roles.
- ▶ There is also a lack of standards around support for best practices in Web development. Specifically, JSF enforces the MVC architecture. Since the inception of Java EE, we have experienced the pain-points of the Model 1 approach. Various Open Source frameworks have tried to address this gap by implementing the MVC architecture. Like all facets of Java EE, a standard based technology helps reduces complexity.

## 8.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed these steps:

- ▶ Defined the two databases and the matching database pools
- ▶ Created the JPA entities and the three session beans
- ▶ Created and run the JUnit test cases

## 8.3 Basic Web application

### Optional!

Creating the complete JSF Web application as described in 8.4, “JSF Web application” on page 297 can take quite a bit of time and might provide more information than needed if your primary focus is in back-end development (JPA, EJB, Web services, or messaging).

Therefore, you can optionally implement this Basic Web application with access to an employee record. This provides you with basic experience with Web development and how to connect a Web application to EJB 3.0 session beans.

However, if you complete this section (and do not complete the JSF Web application), you still must complete or jump start the JSF Web application before starting Chapter 10, “Implement core security” on page 361.

In this section, we create a basic Web application using servlets and JSP. For the purpose of this example, we use out-of-the-box servlet and JSP functionality. Therefore, we do not follow MVC architecture in this example, though at a minimum, we follow some separation of concerns.

Because we are developing two Web applications (one simple and one JSF-based), the approach we take for the simple Web application is to only implement the Find Employee use case.

To further illustrate and clarify our approach, the Web application structure is shown in Figure 8-5 on page 287.

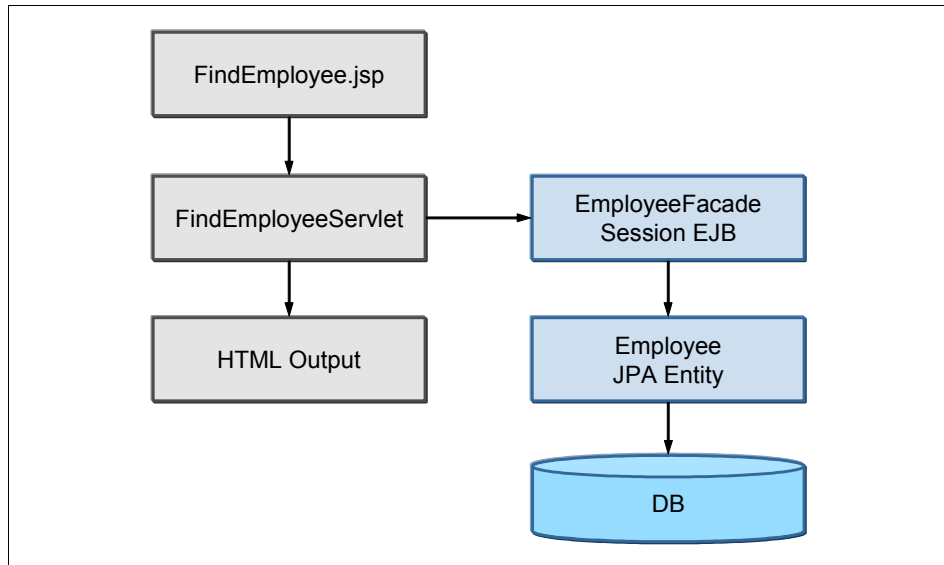


Figure 8-5 Web application structure

### 8.3.1 Create and configure the Web project

To begin this exercise, create a dynamic Web project called **DonateWeb**:

1. In the Java EE perspective Enterprise Explorer, select **File** → **New** → **Dynamic Web Project**.
2. At the Dynamic Web Project pop-up, specify the following:
  - a. Set these values:
    - Project Name: **DonateWeb**.
    - Target IBM Runtime: **IBM WebSphere Application Server v7.0**.
    - Dynamic Web Module version: **2.5**.
    - Configurations: **Default Configurations for IBM WebSphere Application Server v7.0**.
  - b. Select **Add project to an EAR**.
  - c. Set EAR Project Name: **DonateEAR**.
  - d. Click **Next**.

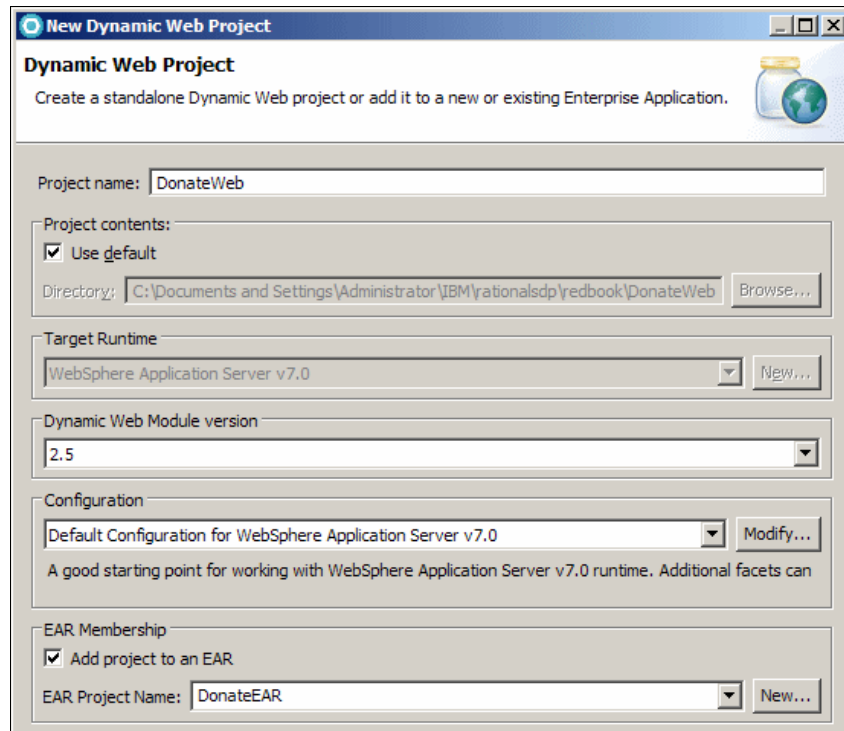


Figure 8-6 Create a dynamic Web project

3. At the Web Module pop-up, select **Generate Deployment Descriptor**, and click **Finish**.
  - a. At the resulting **Open Associated Perspective?** pop-up, click **Yes** to switch to the Web perspective.
4. Optionally review and then close the Technology Quickstarts editor that opened after DonateWeb was created.

### 8.3.2 Add the dependency to the EJB project

In the Web project we access the EJB session beans. Therefore, we have to setup the dependency of the Web project to the EJB project.

By adding the DonateWeb project to the DonateEAR enterprise application, a dependency has been created in DonateEAR. Now we can set up further dependencies between projects contained in DonateEAR:

1. Select **DonateWeb**, right-click and select **Properties**.

- At the Properties pop-up, select **Java EE Module Dependencies**, and select the **DonateEJB.jar** file. Click **OK**.

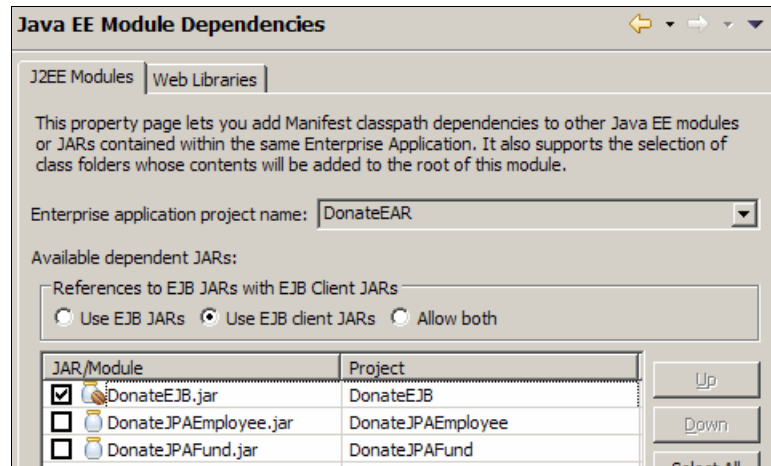


Figure 8-7 Add the Java EE module dependencies

### 8.3.3 Create the JSP

Next, we create the page:

- In the Enterprise Explorer, select **WebContent** (in DonateWeb), right-click and select **New** → **Web Page**.
- At the New JavaServer Page pop-up:
  - Set File Name to **FindEmployee**.
  - Under Template, select **Basic Templates** → **JSP**.
  - Click **Finish**.

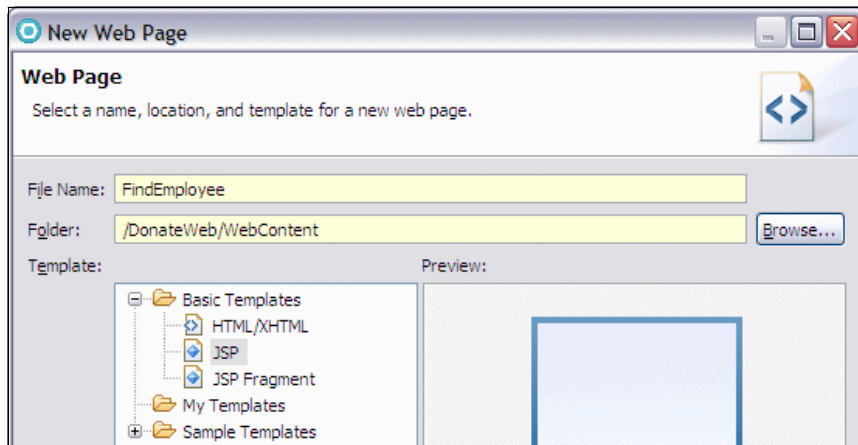


Figure 8-8 New JavaServer Page

3. FindEmployee.jsp opens in the editor (Example 8-1):
  - a. Select the Source tab on the lower part of the editor.
  - b. Insert the **F08.1 Web Front Employee.JSP** snippet into the JSP code between the <body></body> tags.

#### Off Course?

If the Snippets view is not available, open it from the action bar by selecting **Windows** → **Show View** → **Other** and then selecting **General** → **Snippets** from the pop-up. If you do not like the default location of the right pane, then drag and drop to a more suitable location such as the lower central pane (which also contains the Problems, Servers, and Console views).

- c. Save and close FindEmployee.jsp.

#### Example 8-1 F08.1 Web Front Employee JSP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><%@page
  language="java" contentType="text/html; charset=ISO-8859-1"
  pageEncoding="ISO-8859-1"%>
<html>
<head>
<title>FindEmployee</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body>
  → <form action="FindEmployeeServlet" method="post">
```

```
Employee ID:
<input type="text" name="empId" value="" />

<input type="submit" value="Submit" />
</form>

</body>
</html>
```

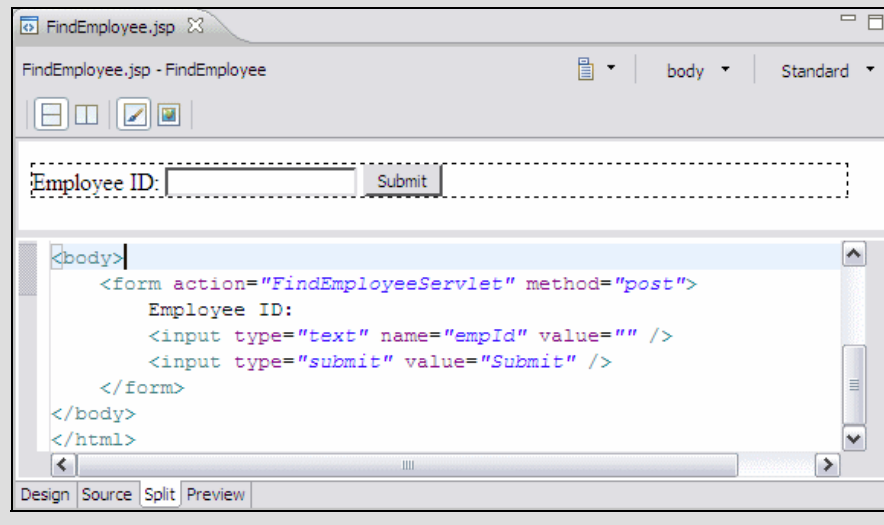
---

### Alternative editor:

Note that you can also open the JSP with a multiple editors. The two most common editors are the JSP editor (which provides a text view with content assistance) and the Page Designer, which was used above.

You used the basic text mode component of Page Designer, but it also has Design, Preview, and Split (text and design) modes.

Note that the last used editor becomes the default for double-click..:



#### Behind the scenes:

Although it is labeled a JSP, `FindEmployee` contains only HTML tags. These tags define an input field that permits the user to enter the employee ID as a text string, and then click **Submit** to redirect the page (with the input value) to the `FindEmployeeServlet` using the post method.

```
<form action="FindEmployeeServlet" method="post">
  Employee ID:
  <input type="text" name="empId" value="" />
  <input type="submit" value="Submit" />
</form>
```

### 8.3.4 Create the servlet

Next we create the servlet, which provides the invocation of the EJB session bean:

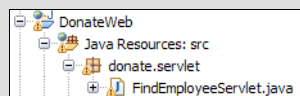
1. In the Enterprise Explorer, select **DonateWeb** project, right-click and select **New** → **Servlet**.
2. At the Create Servlet pop-up, provide the following data:
  - a. Set these values:
    - Java package: **donate.servlet**
    - Class name: **FindEmployeeServlet**
    - Superclass: **Accept javax.servlet.http.HttpServlet**
  - b. Click **Finish** (or go through the dialog pages using **Next** to see other options).

#### Behind the scenes:

In creating a servlet, a servlet class and a corresponding mapping in the `web.xml` are created.

A servlet (like any Java EE component) is a container-managed component. The container must be made aware of the Java EE component. In the case of a servlet, the Web container is made aware of the servlet, and knows what URL is used to invoke the servlet.

- The servlet class is created in the `donate.servlet` package.





- The corresponding entries are inserted into the Web deployment descriptor (DonateWeb/WebContent/WEB-INF/web.xml).

```
<servlet>
  <description></description>
  <display-name>FindEmployeeServlet</display-name>
  <servlet-name>FindEmployeeServlet</servlet-name>
  <servlet-class>donate.servlet.FindEmployeeServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FindEmployeeServlet</servlet-name>
  <url-pattern>/FindEmployeeServlet</url-pattern>
</servlet-mapping>
```

3. Complete the FindEmployeeServlet code that invokes the session bean.
    - a. Delete all the generated method skeletons (default constructor, doGet, and doPost), leaving the base class definition and the serialVersionUID.
- ```
public class FindEmployeeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

}
```
- b. Insert the **F08.2 Web Front Employee Servlet** snippet before the closing brace (Example 8-2).
  - c. Organize imports (**Ctrl+Shift+O**) to resolve:
    - import javax.ejb.EJB;
    - import vacation.ejb.interfaces.EmployeeFacadeInterface;
    - import vacation.entities.Employee;
  - d. Save and close FindEmployeeServlet.java (Example 8-2).

*Example 8-2 F08.2 Web Front EmployeeServlet*

```
public class FindEmployeeServlet extends javax.servlet.http.HttpServlet
                                   implements javax.servlet.Servlet {
    static final long serialVersionUID = 1L;

    @EJB
    EmployeeFacadeInterface employeeFacade;

    protected void doGet(HttpServletRequest request, HttpServletResponse
                           response) throws ServletException, IOException {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
                           response) throws ServletException, IOException {
```

```

String empId = (String) request.getParameter("empId");

Employee obj = null;
try {
    obj = employeeFacade.findEmployeeById(Integer.parseInt(empId));
} catch (Exception e) {
    e.printStackTrace();
}

if (obj == null) {
    response.getWriter().println("null object");
} else {
    response.getWriter().println(formatEmployeeInfo(obj));
}
}

public String formatEmployeeInfo(Employee obj) {
    StringBuffer sb = new StringBuffer();
    sb.append("<html><body>");
    sb.append("employee id = " + obj.getEmployeeId() + " <br><b>");
    sb.append("employee first name = " + obj.getFirstName() + "<br>");
    sb.append("employee middle name = " + obj.getMiddleName() + "<br>");
    sb.append("employee last name = " + obj.getLastName() + "<br>");
    sb.append("Salary = " + obj.getSalary() + " </b></body></html>");
    return sb.toString();
}
}

```

---

### Behind the scenes:

Notice that the code in the servlet is implemented in the **doPost** method. There is a difference in method submissions of HTML forms. The two types are GET and POST:

- **GET:** Request parameters are appended to the URL query string, and the end user can save such a URL with all the parameters.
- **POST:** Request parameters are written in the server input stream, and are not visible to the end user.

The key elements in this servlet are as follows:

- The @EJB annotation provides an instance of the employee facade session bean's business interface (EmployeeFacadeInterface). This is the injection technique of EJB 3.0, much simpler than using JNDI.

@EJB

EmployeeFacadeInterface employeeFacade;

- The application code required is the same for both a doPost and a doGet, so we redirect all doGet requests to the doPost method.


```
protected void doGet(HttpServletRequest request, HttpServletResponse  
                    response) throws ServletException, IOException {  
    doPost(request, response);  
}
```





- The doPost method uses the input parameter to look up the employee record from the EmployeeFacade session facade, and then calls the formatEmployeeInfo method (not shown here) to format the result.

```
protected void doPost(HttpServletRequest request, HttpServletResponse  
                    response) throws ServletException, IOException {  
  
    String empId = (String) request.getParameter("empId");  
    Employee employee = null;  
    try {  
        employee = employeeFacade.findEmployeeById  
                                (Integer.parseInt(empId));  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    if (employee == null) {  
        response.getWriter().println("Employee not found");  
    } else {  
        response.getWriter().println(formatEmployeeInfo(employee));  
    }  
}
```

### 8.3.5 Test the Web application

To test the Web application, we deploy it to the server and run the servlet:

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to:
    -  C:\IBM\SDP\runtimes\base\_v7\derby\bin\networkServer

-  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`  
and execute:
  -  `startNetworkServer.bat`
  -  `startNetworkServers.sh`
- If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
- If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.

#### Behind the scenes:

Recall that you started both of these in earlier chapters. They should still be running unless you stopped them or restarted the system.

- ▶ 4.2, “Start the Derby Network Server” on page 87.
- ▶ 3.6, “Start the ExperienceJEE Test Server” on page 78.

2. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
3. Test the code. Select **FindEmployee.jsp** (in DonateWeb/WebContent), right-click and select **Run as** → **Run on Server**
  - a. At the Run On Server pop-up. Select **localhost** → **ExperienceJEE Test Server**.
  - b. Enable Always use this server when running this project.
  - c. Click **Finish**.

**Behind the scenes:** you can also change this setting from the Servers tab on the DonateWeb properties pop-up.

The initial Web page is displayed.

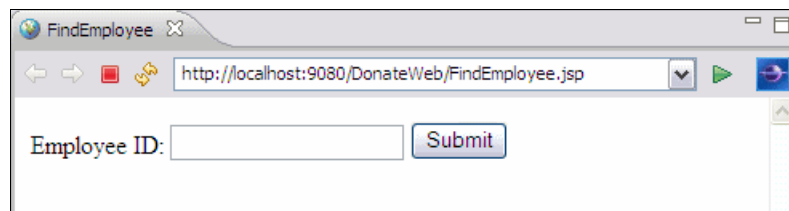


Figure 8-9 Initial Web page

4. In the Employee ID field, type the number **1** and click **Submit**.  
The employee details page is displayed.

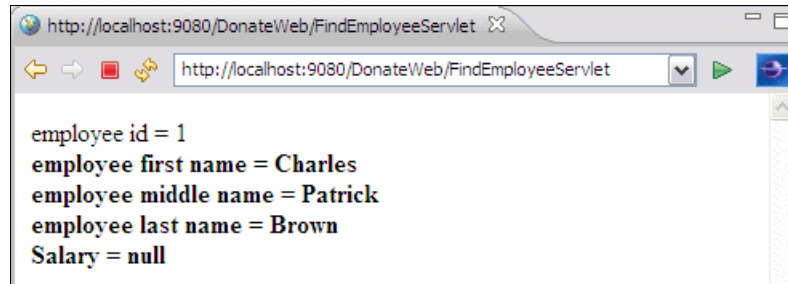


Figure 8-10 Employee details results

5. You can also try employee ID 2 or 3. Any other ID gets an error message.

### 8.3.6 Recap

In this section, we have gained practical experience in developing a simple Web-based application using JSP and servlet technology. Because we were using out-of-the-box functionality from JSP and servlet technology, we only partially followed the Model 2 approach. We did not have a clear separation of concerns between the view and the model layers.

As the Java EE specifications evolved, the adoption of full Model 2 (MVC) paradigm (in addition to Web component-based programming) was eventually included in the specification. These features were included in the JavaServer Faces (JSF) specification. For more information about JSF and its capabilities, please read the rest of this chapter.

## 8.4 JSF Web application

In the remainder of this chapter we develop a JSF Web application with multiple pages.

We are developing an application that facilitates the donation of vacation days to employees who are in need of extra vacation days. The following are the use cases that we implement for this application:

- ▶ **Employee search:** A page where employee id is provided.
- ▶ **Employee detail:** A page where employee details are displayed.

- **Vacation donation:** A page where vacation donation can be specified

Taking a further step, we build a JSF-based front end to the Donate application by implementing the following pages:

- **Find an employee:** This page will provide the search functionality for an employee by employee ID.
- **Employee details:** This page displays the employee details. The page also contains a form where vacation hours donation can be specified and submitted.
- **Donation confirmation:** This page confirms the vacation hours donation specified in the employee details page.

### 8.4.1 Suppress page code generation

By default the Rational Application Developer tooling automatically creates page-specific Faces managed beans with each JSP called page code beans. This can be simplify the development of basic applications by eliminating the necessity of manually creating managed beans.

In addition, some IBM enhanced Faces components such as relational records and relational record lists require page code managed beans, as does custom validation as described in 8.8.2, “Validation” on page 342.

However, in this case, since you will be using standard Faces components and creating specific managed beans to share across the pages, the page code managed beans would simply be additional unused artifacts.

Refer to the Rational Application Developer Information Center *Viewing page code for a Faces JSP file* topic for further information:

<http://publib.boulder.ibm.com/infocenter/rsawshlp/v7r5m0/index.jsp?topic=/com.ibm.etools.jsf.doc/topics/cjsffacets.html>

1. In the workbench action bar, select **Window** → **Preferences**.
2. At the Preferences pop-up:
  - a. On the left, select **Web** → **JavaServer Faces Tools** → **Page Code**.
  - b. On the right (in the resulting Page Code), enable Suppress Page Code file generation.
  - c. Click **OK**.

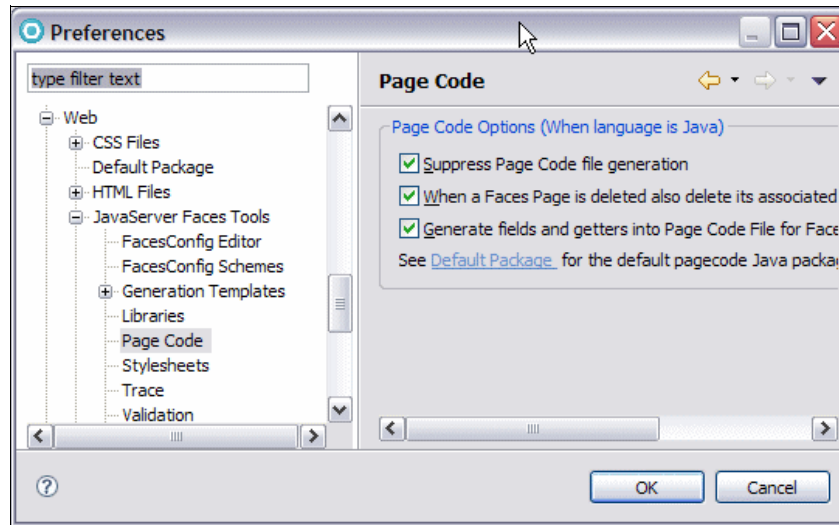


Figure 8-11 Suppressing page code file generation in the preferences

### 8.4.2 Create the JSF Web project

We create a Dynamic Web project named **DonateJSFWeb** for JSF development, and add the Web project to the DonateEAR enterprise application.

1. In the Java EE perspective, Enterprise Explorer, select **File** → **New** → **Dynamic Web Project**.
2. At the Dynamic Web Project pop-up, specify the following parameters:
  - Project Name: **DonateJSFWeb**
  - Target IBM Runtime: **IBM WebSphere Application Server v7.0**
  - Dynamic Web Module version: **2.5**
  - a. Under Configuration, select **JavaServer Faces IBM Enhanced Project** and click **Modify**.
  - b. At the resulting Project Facets pop-up, clear the **JavaServer Faces (IBM Enhanced)** selection and click **OK**.

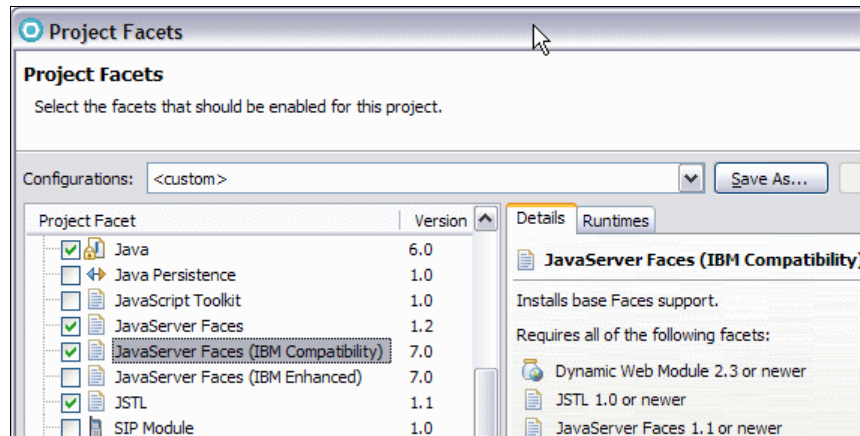


Figure 8-12 Clear the JavaServer Faces (IBM Enhanced) selection

### Behind the scenes:

Many vendors, including IBM, support both the standard JavaServer Faces specification and vendor specific enhancements. For example, standard Faces components do not support interaction with AJAX components where-as the IBM Enhanced components do support this interaction.

By deselecting the IBM Enhanced Facet you are making the choice adhere to the basic specification.

Refer to the Rational Application Developer Information Center *JSF Facets* topic for further information:

<http://publib.boulder.ibm.com/infocenter/rsawshlp/v7r5m0/topic/com.ibm.etools.jsf.doc/topics/cjsffacets.html>

3. Back at the Dynamic Web Project pop-up, note that the Configuration is now set to <custom>. Under EAR Membership:
  - a. Enable **Add project to an EAR**.
  - b. Set the EAR Project Name to **DonateEAR**.
  - c. Click **Next**.



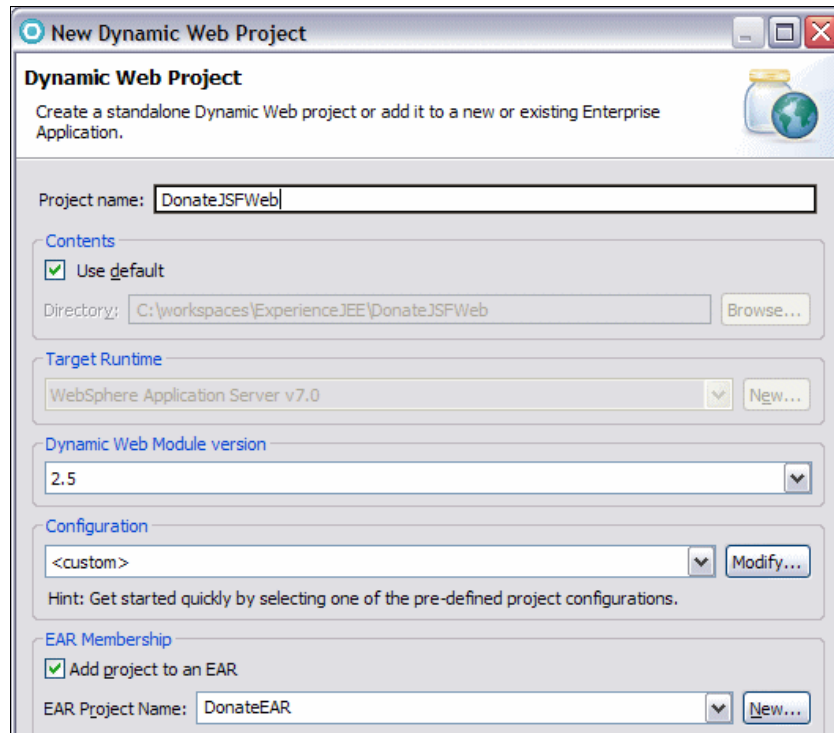


Figure 8-13 Dynamic Web project settings

4. At the Web Module pop-up, select **Generate Deployment Descriptor**, and click **Next**.
5. At the JSF Capabilities pop-up, select **Server Supplied JSF Implementation** and click **Finish**.

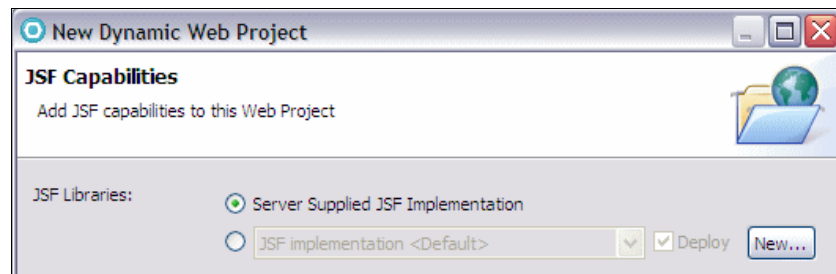


Figure 8-14 Select the JSF capabilities

6. Optionally review and then close the Technology Quickstarts editor that opened after DonateJSFWeb was created.

### Behind the scenes:

The layout of the JSF Web project is the same as for the simple Web project, with these additions:

- ▶ A Faces configuration file (WEB-INF/faces-config.xml) has been created. The Faces configuration file contains information about navigation between pages, validators, and managed beans. You can open the file, but it is basically empty now.
- ▶ A Faces servlet is predefined. The Faces servlet handles all the requests. You can see the servlet in the Web deployment descriptor (WEB-INF/web.xml).
- ▶ The URL mapping of **/faces/\*** or **/\*.faces** directs all such requests to the Faces servlet. The equivalent URLs are:

```
http://<host>:9080/<contextroot>/faces/xxxx.jsp  
http://<host>:9080/<contextroot>/xxxx.faces
```

This URL mapping is controlled by the servlet mapping definitions in the Web deployment configuration file:

```
<servlet-mapping>  
    <servlet-name>Faces Servlet</servlet-name>  
    <url-pattern>  
        *.faces</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
    <servlet-name>Faces Servlet</servlet-name>  
    <url-pattern>  
        /faces/*</url-pattern>  
</servlet-mapping>
```

Attempting to access the standard non-Faces URL notation (http://<host>:9080/<contextroot>/xxxx.jsp) will result in the following error because the JSP is called before the Faces context is initialized:

```
Error 500: java.lang.RuntimeException: Cannot find FacesContext
```

## 8.4.3 Create the JSPs

The approach that we take to develop the JSF front end is page by page, from the finding an employee page (FindEmployee.jsp), to the employee details page (EmployeeDetails.jsp), to the donation confirmation page (DonationConfirmation.jsp).

First we create the empty JSPs:

1. Select **WebContent** (in DonateJSFWeb), right-click and select **New** → **Web Page**.
2. At the New JavaServer Page pop-up:
  - Set File Name to **FindEmployee**.
  - Under Template, select **Basic Templates** → **JSP**.
3. Click **Finish**.
  - Review the resulting FindEmployee.jsp editor and close when finished.

#### Behind the scenes:

Note the JSF tag library that is added to the JSP source code. This will be used to reference the JSF statements that will be added in 8.6.1, “Implement the FindEmployee page” on page 318.

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
```

4. Repeat this sequence for EmployeeDetails.jsp.
5. Repeat this sequence for DonationConfirmation.jsp.

### 8.4.4 Define the navigation

In servlet programming, we do not have any provisions on how to specify navigation throughout an application. Typically, the navigation through an application is implemented programmatically within the servlet.

In JSF, we define navigation declaratively through an external configuration file. The advantage of declarative programming is that page navigation is externalized from code (easily accessible), which eases development and maintenance:

1. Open **faces-config.xml** (in DonateJSFWeb/WebContent/WEB-INF), and select the **Navigation Rule** tab. On the far right-hand side, select **Palette** and select the **Nodes** → **Page** icon.

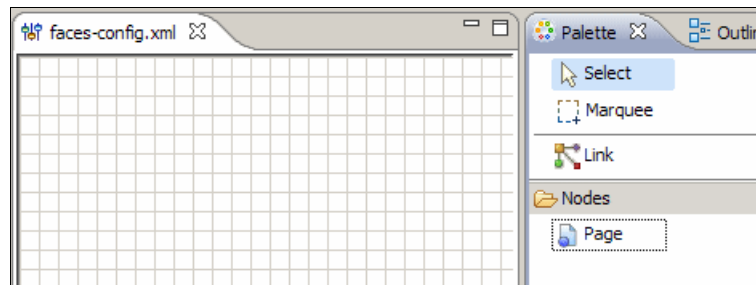
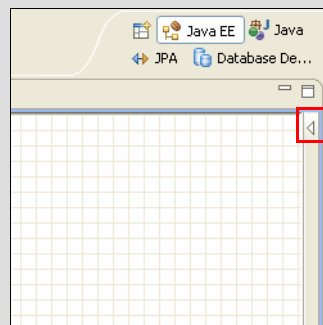


Figure 8-15 Define the navigation - step 1

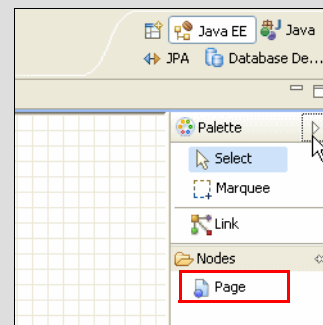
### Off course?

If you do not see the Palette, it is probably minimized. Look for the left facing arrow on the upper right side of the faces-config.xml editor and click the arrow to open the palette.

#### ► Hidden palette:



#### ► Visible palette:



2. Select the **Page** icon and click into the empty space in the faces-config.xml editor.

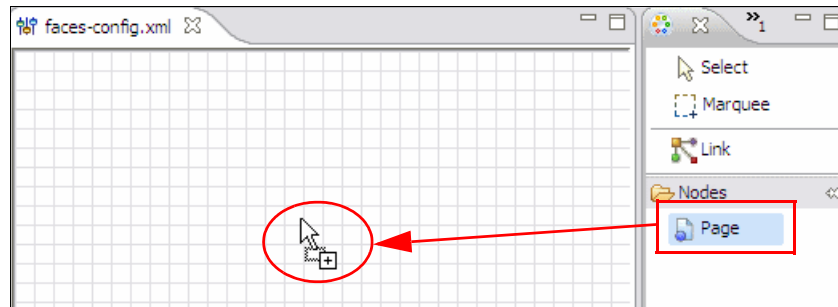


Figure 8-16 Define the navigation - step 2

3. At the resulting Select JSP File pop-up:
  - a. Expand **DonateJSFWeb** → **WebContent** and select **FindEmployee.jsp**.
  - b. Click **OK** and the FindEmployee page is added to the diagram.

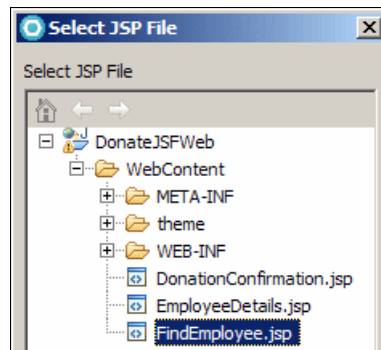


Figure 8-17 Define the navigation - step 3

4. Repeat these steps to add EmployeeDetails.jsp
5. Repeat these steps to add DonationConfirmation.jsp

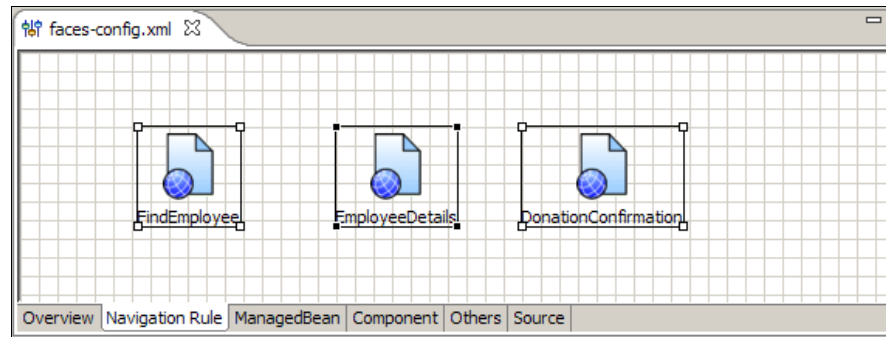


Figure 8-18 Define the navigation - step 4

6. Create the navigation links between the pages:
  - a. In the **Palette**, select the **Link** icon.
  - b. Click on **FindEmployee** and drag the link to **Employee Details**. A link line is drawn between the pages.
  - c. Repeat this and draw a link from **EmployeeDetails** to **DonationConfirmation**

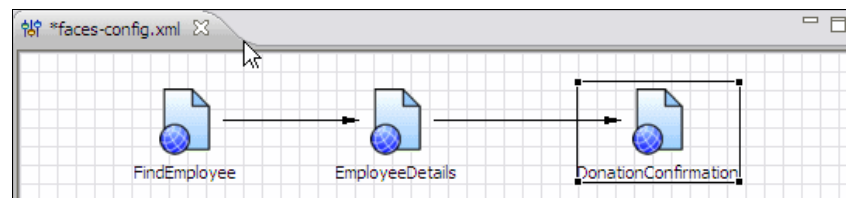


Figure 8-19 Define the navigation - step 5

7. Configure each navigation link to set the navigation and the label. To do this, you will use three separate views as shown in Figure 8-20:

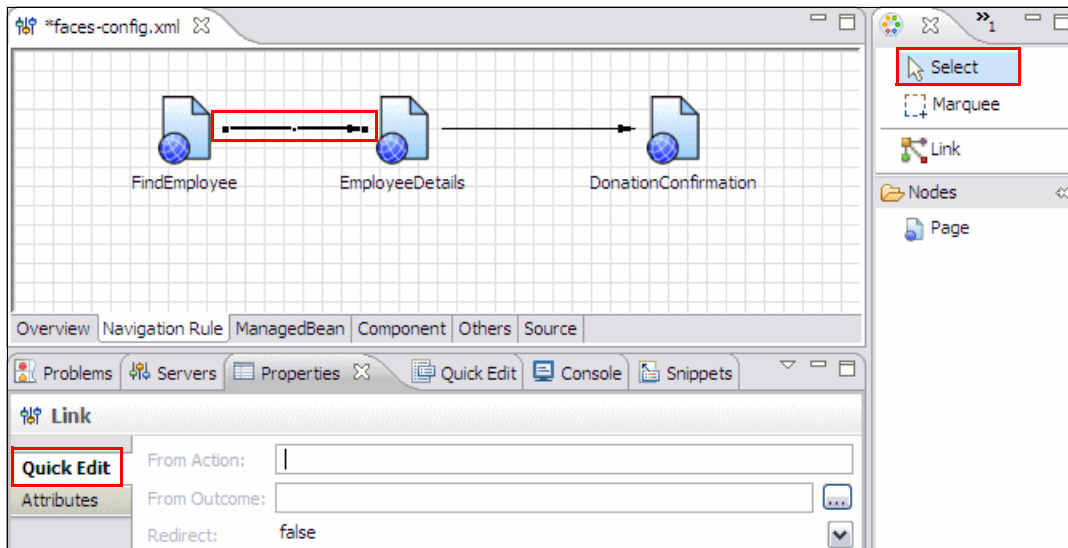


Figure 8-20 Define the navigation - step 6

- a. In the Palette in the upper right, select the **Select** icon.
- b. In the faces-config.xml editor, right-click the navigational link from FindEmployee to EmployeeDetails.
- c. In the lower central pane, switch to the Properties view. If the Properties view is not available, select **Show View** → **Properties** to open the view).
  - i. Select the **Quick Edit** tab:
  - ii. Set the From Outcome to **success**.
  - iii. Set Redirect to **true** using the pull-down menu.

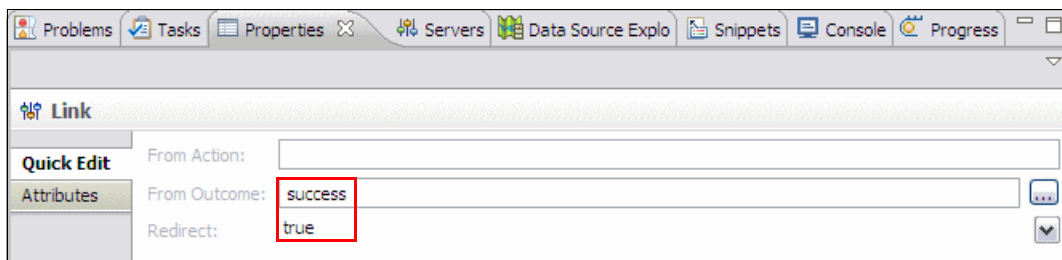


Figure 8-21 Define the navigation - step 7

- d. Repeat this for the link from EmployeeDetails to DonationConfirmation (using **success** for the From Outcome and leave Redirect as **false**). The Navigation Rule page shows the navigation names.

### Behind the scenes:

Refer to the Behind the scenes box (on the next page) and to 10.5.2, “Restrict access to Web pages via declarative security” on page 375, for a description of why we set redirect to true for the first link, but to false for the second link.

8. View the updated content in both the Navigation Rule tab (graphical view) and in the Source tab (Example 8-3).
  - a. Save and faces-config.xml when done but leave open because you will make further changes in the next section.

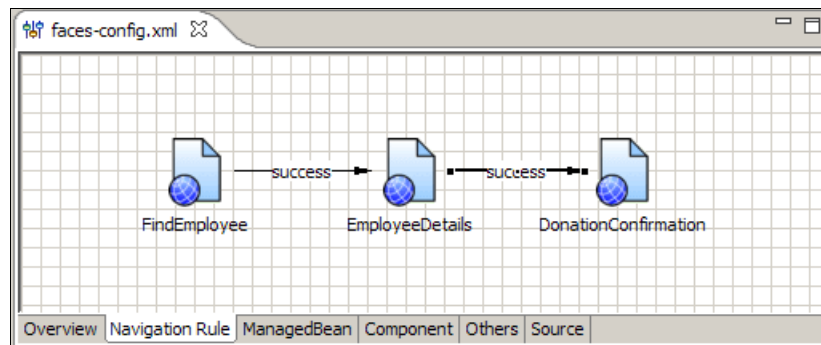


Figure 8-22 Define the navigation - step 8

### Example 8-3 Navigation rules in faces-config.xml

```
<navigation-rule>
  <display-name>FindEmployee</display-name>
  <from-view-id>/FindEmployee.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/EmployeeDetails.jsp</to-view-id>
    <redirect />
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <display-name>EmployeeDetails</display-name>
  <from-view-id>/EmployeeDetails.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/DonationConfirmation.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```



### Behind the scenes:

- The XML element **<from-view-id>** represents the current page. The element **<from-outcome>** represents the indicator of the outcome of the invocation of the business logic. The element **<to-view-id>** represents the page to forward or redirect to based on the outcome of the invocation of the business logic.

In our example this means: If the `FindEmployee.jsp` returns the outcome **success**, then the `EmployeeDetails.jsp` is invoked.

- **Request redirection** is used instead of **request forwarding** to ensure that Java EE role-based security can be applied to unique pages.

```
<to-view-id>  
/EmployeeDetails.jsp</to-view-id>  
<redirect/>
```

Java EE role-based security for Web artifacts are controlled at the URI level:

- Request forwarding controls the page navigation on the server side, so the URI submitted from the client side does not change, and would therefore restrict us to a single security constraint (targeted for `DonateJSFWeb/FindEmployee.jsp`) but applying to all subsequent pages accessed through request forwarding.
  - Request redirection passes the new URI back to the client side, and therefore allows separate security constraints for each page.
- So, why use request forwarding and why is it the default? The answer is to prevent the user of the Web application from knowing the URLs of secondary pages (`EmployeeDetails` or `DonationConfirmation`). This ensures that they always start at the primary entry point into the Web application (`FindEmployee.jsp`).

With request forwarding, the user can see the secondary Web pages and can bookmark one of them. What happens when the user reloads that bookmark (after the session information has expired)? The Faces Managed bean (which we create in the next section) will no longer have a value, and therefore you must put extra logic into the Web application to handle this condition of incorrect page navigation.

- ▶ The choice presented so far is not pleasant: Use request forwarding (to allow multiple Java EE declarative role-based security policies but allow the potential for incorrect page navigation, or use request redirection and limit Java EE declarative role-based security to a single policy for the entire Web application, but reduce the potential for incorrect page navigation.
- ▶ There is a third approach: Stick with request forwarding (and a single Java EE declarative role-based security policy), but then use Java EE programmatic role-based security to control the transition from one page to another. See 10.6.4, “Implement programmatic security in a Web application” on page 397 for more details on programmatic security.

## 8.5 Create the managed beans

In this section, we create the managed beans that hold the data and action code for the pages. We use a package named **donate.jsf.bean** to hold two managed beans:

- ▶ **FindEmployeeManagedBean**: Holds the employee data and the action logic to find an employee.
- ▶ **DonateHoursManagedBean**: Holds the data and action logic to donate hours to the specified donation fund.

### 8.5.1 Define the managed beans

Managed beans are defined in the Faces configuration file. They are managed (that is, allocated) at execution time when needed for pages:

1. Switch to the open faces-config.xml editor. If it is not open, open **faces-config.xml** (in DonateJSFWeb/WebContent/WEB-INF).

2. Select the **ManagedBean** tab and click **Add**.

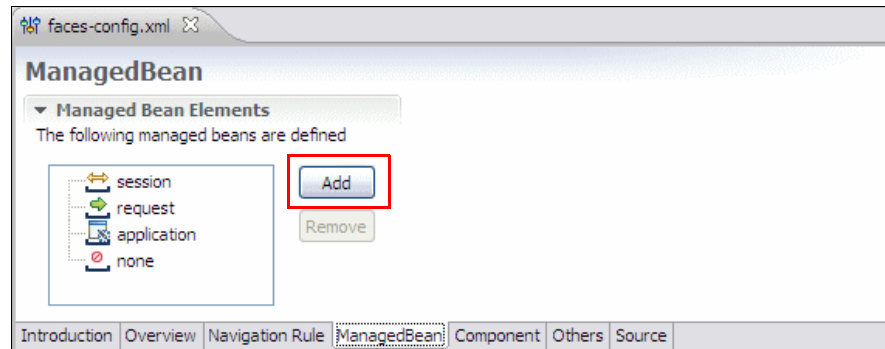
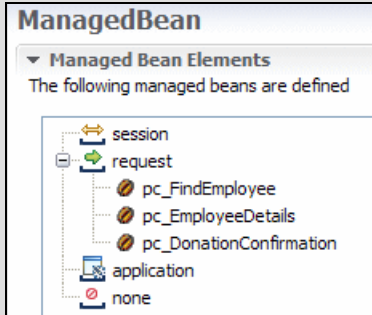


Figure 8-23 ManagedBean tab



**Off Course?:**

If you see three existing pc\_\* request beans then you neglected to suppress the page code generation in 8.4.1, “Suppress page code generation” on page 298.

At this point, leave the page code beans as defined. They will simply be additional artifacts that are not used in this scenario.

3. At the New Managed Bean Wizard/Java Class Selection pop-up, select **Create a new Java class**, and click **Next**.
4. At the New Managed Bean Wizard/Java Class pop-up:
  - a. Set Package to **donate.jsf.bean**.
  - b. Set Name to **FindEmployeeManagedBean**.
  - c. Click **Next**.

5. At the Managed Bean Configuration pop-up, accept the name, and for Scope select **session**. Click **Finish**.

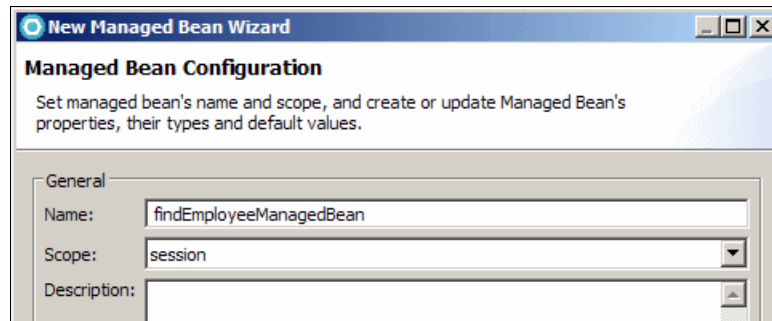


Figure 8-24 Managed bean configuration

6. Repeat this sequence for the **DonateHoursManagedBean** class, but set the scope to **request**.
7. Save faces-config.xml, but do not close it.

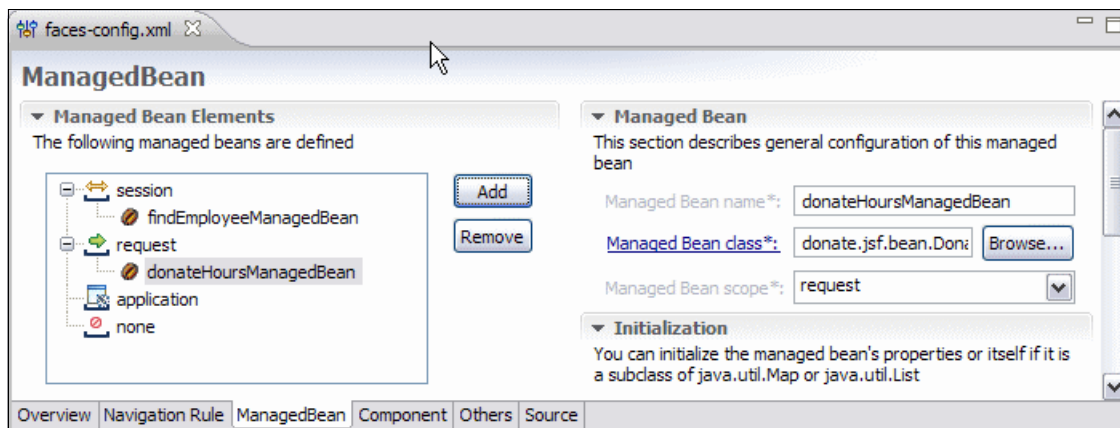


Figure 8-25 faces-config.xml

8. Select the **Source** tab. Notice that the faces-config.xml has two mappings for managed beans).
- a. Save and close faces-config.xml

```
<managed-bean>
  <managed-bean-name>findEmployeeManagedBean</managed-bean-name>
  <managed-bean-class>donate.jsf.bean.FindEmployeeManagedBean</man...>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

```
<managed-bean>
  <managed-bean-name>donateHoursManagedBean</managed-bean-name>
  <managed-bean-class>donate.jsf.bean.DonateHoursManagedBean</man...>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

---

### Behind the scenes:

Notice that two managed beans were created. One managed bean resides in request scope. The other managed bean resides in session scope:

- ▶ Request scope means that all attributes and parameters are stored in the HTTP request and are only good for one HTTP request/response pair.
- ▶ Session scope means that all attributes and parameters are stored in the HTTP session and persist until explicitly removed.

So, when do we store values in session versus request? Ideally, most values should live in the request scope. Values that might be expensive to re-create and that are needed across pages within an application can be stored in the session scope. However, excess storage of objects in the session scope causes session bloat, which can negatively influence application performance.

## 8.5.2 Add the dependency to the EJB project

Because the JSF managed beans require access to the classes of the session EJBs, we have to create a dependency of the `DonateJSFWeb` project to the `DonateEJB` project:

1. In the Enterprise Explorer, select **DonateJSFWeb**, right-click and select **Properties**.
2. At the Properties pop-up:
  - a. Select **Java EE Module Dependencies** on the left.
  - b. Enable the **DonateEJB.jar** module on the right.
  - c. Click **OK**.

## 8.5.3 Implement the `FindEmployeeManagedBean` class

In 8.6.2, “Bind `FindEmployee` to a managed bean” on page 323, we will bind fields in `FindEmployeeManagedBean` for an input field and a Submit button to this managed bean:

```
Input field:    value="#{findEmployeeManagedBean.empId}"
```

```
Submit button: action="#{findEmployeeManagedBean.findEmployee}"
```

Therefore, we have to define an `empId` property to hold the input field value, and a `findEmployee` method to hold the action code to access the `EmployeeFacade` session bean and retrieve the employee:

1. Open **FindEmployeeManagedBean** (in `DonateJSFWeb/Java Resources: src/donate.jsf.bean`).
2. Place the cursor before the closing brace and insert these two variables. You can ignore for now the resolution error for `Employee` because you will organize imports after making several additional changes.

```
private String empId;  
private Employee employee;
```

3. Select the source code, right-click and select **Source** → **Generate Getters and Setters**.
  - Select both properties and click **OK**. This creates the get and set methods for the two properties.
4. To access the `EmployeeFacade` session bean, add the following statement after the class definition:

```
public class FindEmployeeManagedBean {  
    @EJB EmployeeFacadeInterface employeeFacade;  
  
    private String empId;
```

5. Create the `findEmployee` method by inserting the **F08.3 JSF Employee Managed Bean findEmployee** snippet from the Snippets view at the end of the class before the closing brace (Example 8-4).

*Example 8-4 F08.3 JSF Employee Managed Bean findEmployee*

---

```
public String findEmployee() {  
    Employee employee = null;  
    try {  
        employee = employeeFacade  
            .findEmployeeById(Integer.parseInt(getEmpId()));  
        setEmployee(employee);  
        return "success";  
    }  
    catch(Exception e) {  
        e.printStackTrace();  
    }  
    return ""; // stay on the same page  
}
```

---

6. Organize imports (**Ctrl+Shift+O**) to resolve:

```
javax.ejb.EJB;
```

```
import vacation.ejb.interfaces.EmployeeFacadeInterface;
import vacation.entities.Employee;
```

#### 7. Save and close FindEmployeeManagedBean.java

##### Behind the scenes:

- ▶ The findEmployee method takes the value from the empId property, which is mapped to the input field in the FindEmployee JSP.
- ▶ The method then uses the EmployeeFacade session bean to retrieve the employee, and stores the employee.
- ▶ Finally, the method returns success to pass control to the EmployeeDetails JSP.

Remember that the whole bean is stored in session scope, therefore, the employee is available to the next JSP.

### 8.5.4 Implement the DonateHoursManagedBean class

The DonateHoursManagedBean is used for the logic to donate hours from an employee to a fund. Therefore, we have to define an employee property to hold the employee data, and a donateHours method to hold the action code to access the DonateBean session bean and perform the donation of vacation hours:

1. Open **DonateHoursManagedBean** (in DonateJSFWeb/Java Resources: src/donate.jsf.bean).
2. Place the cursor before the closing brace and create three properties and the name of the default fund. You can ignore for now the resolution error for Employee because you will organize imports after making several additional changes.

```
private Employee employee;
private final String fundname = "DonationFund";
private int hoursDonated;
private String resultMessage;
```

3. Select in the source code, right-click and select **Source** → **Generate Getters and Setters**.

- Select resultMessage, hoursDonated, and employee, and click **OK**. Do not select fundname.

4. To access the EmployeeFacade and DonateBean session beans, add the following statements after the class definition

```
public class DonateHoursManagedBean {
    @EJB DonateBeanRemote donateBean;
    @EJB EmployeeFacadeInterface employeeFacade;
```

**Behind the scenes:** Note that we have to use the remote interface of the DonateBean session bean, because the business interface does not have the donateToFund method that returns a result.

5. Create the donateHours method by inserting the **F08.4 JSF Donate Managed Bean donateHours** snippet from the Snippets view at the end of the class before the closing brace (Example 8-5).

*Example 8-5 F08.4 JSF Donate Managed Bean donateHours*

---

```
public String donateHours() {
    FacesContext ctx = FacesContext.getCurrentInstance();
    HttpServletRequest request = (HttpServletRequest)
        ctx.getExternalContext().getRequest();
    FindEmployeeManagedBean employeeManagedBean =
        (FindEmployeeManagedBean) request.getSession()
            .getAttribute("findEmployeeManagedBean");
    String empId = employeeManagedBean.getEmpId();
    try {
        int employeeId = Integer.parseInt(empId);
        String msg = donateBean.donateToFund(employeeId, fundname,
            getHoursDonated());

        setResultMessage(msg);
        setEmployee(employeeFacade.findEmployeeById(employeeId));
        return "success";
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "";
}
```

---

6. Organize imports (**Ctrl+Shift+O**) to resolve:

```
import javax.ejb.EJB;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;
import vacation.ejb.interfaces.EmployeeFacadeInterface;
import vacation.entities.Employee;
import donate.ejb.interfaces.DonateBeanRemote;
```

7. Save and close DonateHoursManagedBean.java.



**Behind the scenes:**

- ▶ The FacesContext is a standard JSF class used to retrieve elements of the page, such as the request block.
- ▶ From the request block we retrieve the session data (the managed bean).
- ▶ We execute donateToFund in the DonateBean session bean to donate a number of hours from an employee to the default fund.
- ▶ We store the result in a property, that will be mapped to the page.
- ▶ We retrieve the employee again to get the remaining vacation hours.

**Recapitulation:**

Consider what we have done for the business logic:

- ▶ We created and implemented two managed beans and configured them in faces-config.xml, including their scope (session or request).
- ▶ We provided the properties to hold input and output values of the JSPs.
- ▶ We implemented the controller logic for each action.

So how is this different from servlet programming? In servlet programming:

- ▶ The equivalent of the managed beans are the servlets. Servlets are configured in the web.xml. The servlets contain the controller logic.
- ▶ Managed beans have to declaratively specify scope, because they contain form data that JSF automatically retrieves and sets in either request or session. In servlet programming, retrieval of form data from either request or session is done programmatically.
- ▶ The navigation across a Web application is also declarative in JSF. We got a taste of how this is achieved when we configured the navigation rules in faces-config.xml. In servlets, navigation across a Web application is programmatic.

## 8.6 Implement the JSP pages

In this section we complete the pages that were defined in 8.4.3, “Create the JSPs” on page 302.

For reference, the final code of the pages is available in the Snippets view:

- ▶ F08.5 JSF Find Employee JSP
- ▶ F08.6 JSF Employee Details JSP
- ▶ F08.7 JSF Donation Confirmation JSP

### 8.6.1 Implement the FindEmployee page

For the `FindEmployee.jsp`, we use the JSF Palette to build the JSP. The JSP has three widgets on the page:

- ▶ Input field for the employee ID (Text Input component).
- ▶ Label of the input field (Output Text component).
- ▶ Submit button to invoke the execution of the employee search function (Command Button component).

To implement the `FindEmployee` page, follow these steps:

1. Select **FindEmployee.jsp** (in `DonateJSFWeb/WebContent`), right-click and select **Open With** → **Page Designer**.
2. Ensure that you can locate the various views that will be used in the subsequent steps:
  - a. The `FindEmployee.jsp` editor is in the upper central pane. Ensure that the Split tab is selected with a design view on top and the source view below. The Palette view is in the right-most pane (it may be hidden but you can select it from the pull down selection).
  - b. Click **Standard Faces Components** to expand that category of components. The Properties view is in the lower central pane.

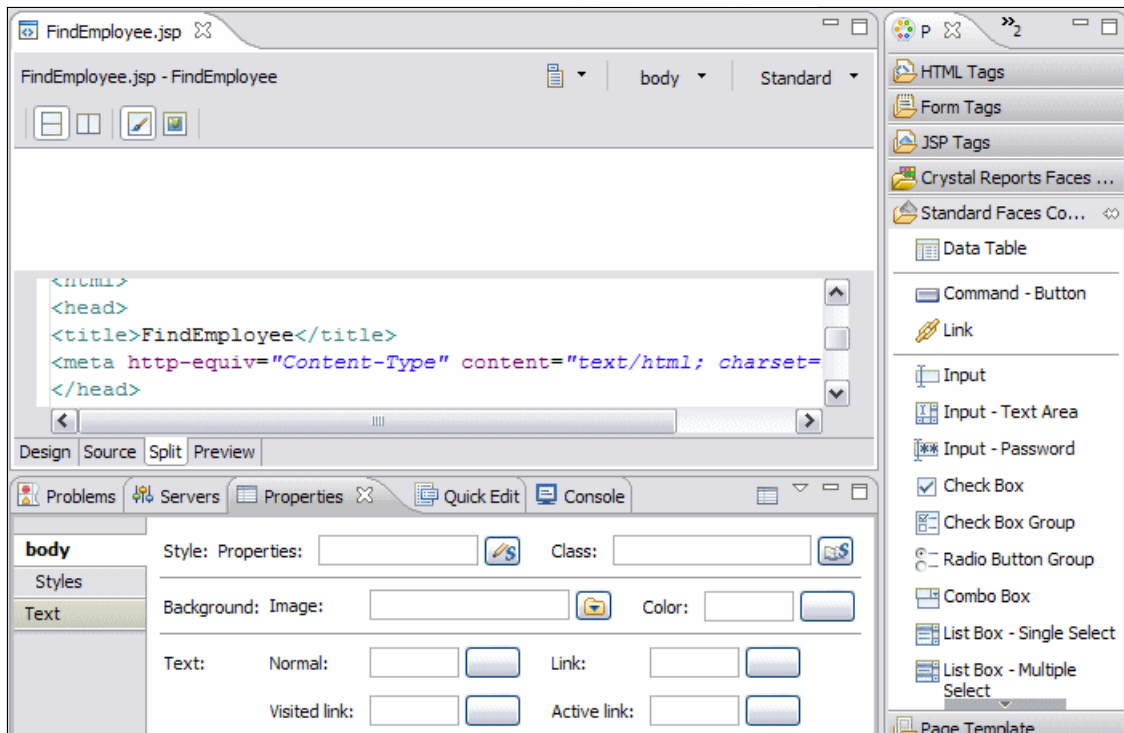


Figure 8-26 FindEmployee.jsp open in the Page Designer

**Off Course:** If you instead see Enhanced Faces Components, then you neglected to clear the JavaServer Faces (IBM Enhanced) facet in 8.4.2, “Create the JSF Web project” on page 299.

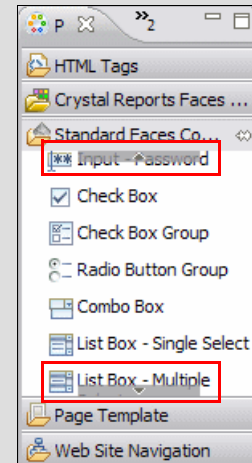
You can resolve this by following these steps:

- ▶ Close faces-config.xml.
- ▶ In the Enterprise Explorer, select **DonateJSFWeb**, right-click and select **Properties**.
- ▶ At the resulting Properties pop-up:
  - On the left select **Project Facets**.
  - On the right clear **JavaServer Faces (IBM Enhanced)**.
  - Click **OK**.
- ▶ Re-open **faces-config.xml**. The Palette view should now show the Standard Faces Components category.

3. Select the **Output** component and drag it into the page into the upper preview pane rectangle in the editor (which is the <f:view> tag inside the <body>).

### Off course?

You may need to scroll down to see the Output component by using the scroll up and scroll down icons at each end of the expanded category.



4. In the Properties view, set the Value field to **Employee ID:** (with a trailing space).

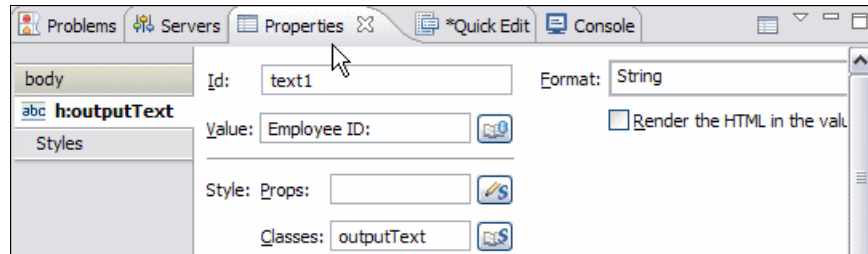


Figure 8-27 Set Value field

### Behind the scenes:

The `h:outputText` tab in the Properties view is an indicator that you selected the desired component in the editor design pane.

Note that the change in the Properties view is automatically reflected in the editor:

#### ► Design pane:



#### ► Source pane;

```
<h:outputText styleClass="outputText" id="text1" value="Employee ID:
"></h:outputText>
```

5. Drag an **Input** component below the output text.

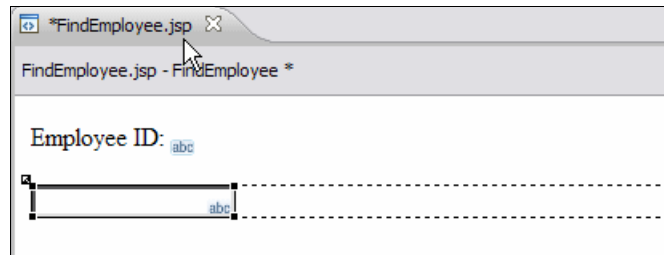


Figure 8-28 Add an input component

6. Drag the Employee ID output component to the beginning of the Input component. Drop the output component onto the left edge of the dotted box.

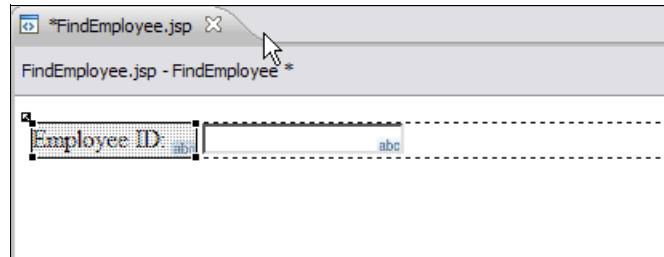


Figure 8-29 Add an output component

7. Select the input text field.
  - a. In the Properties view **h:inputText** tab set Id to **empId**.

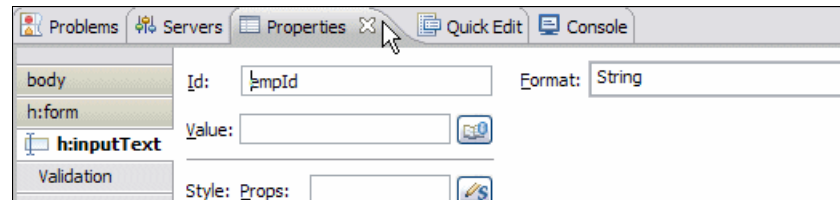


Figure 8-30 Input properties

In the editor source pane, observe the updated ID and add statements for maximum length and size, so that the tag becomes:

```
<h:inputText id="empId" maxLength="6" size="10"></h:inputText>
```

8. Drag a **Command Button** and drop into the form (the dotted box) to the right of the input field.
  - a. In the Properties view **h:commandButton** tab, set Id to **submit**.

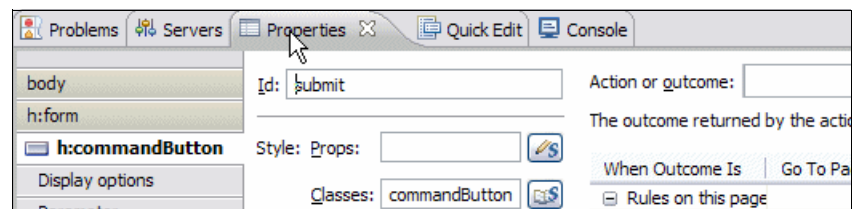


Figure 8-31 Command button properties

- b. In the editor source pane, observe the updated statement:

```
<h:commandButton type="submit" value="Submit" label="Submit"
  styleClass="commandButton" id="submit"></h:commandButton>
```

### Behind the scenes:

The Properties view provides a formatted representation of the source statements across multiple screens. As a result, sometimes it can be difficult to identify the exact correlation between the source view and the Properties view.

In the above case, the following

<u>Source Statement</u>	<u>Properties View equivalent</u>
<code>type="submit"</code>	Base tab, Type option
<code>value="Submit"</code>	No equivalent
<code>label="Submit"</code>	Display options tab, Button label
<code>styleClass="commandButton"</code>	Base tab, Style/Classes
<code>id="submit"</code>	Base tab, Id

9. In the editor source pane, set the title tag:

to `<title>Donate - Search</title>`.

10. Save `FindEmployee.jsp` but leave the editor open because you will make further changes in the next section.

## 8.6.2 Bind `FindEmployee` to a managed bean

The `FindEmployee` page stores the employee ID into the managed bean named `FindEmployeeManagedBean`, and the Submit button executes a method in that JavaBean. We defined this bean in 8.5.1, "Define the managed beans" on page 310.

### Behind the scenes:

The managed beans do not have to exist prior to defining the binding in the JSP file. However, if a referenced managed bean does not exist, you will see a warnings in the Problems view:

```
findEmployeeManagedBean cannot be resolved
```

Therefore, in general it is considered to be a best practice to define the managed beans before completing the JSPs. However, in some situations you might find it appropriate to complete the JSP definitions first.

Follow these steps:

1. In the editor design pane, select the input field in the JSP, and in the Properties view click the browse button next to the Value field.

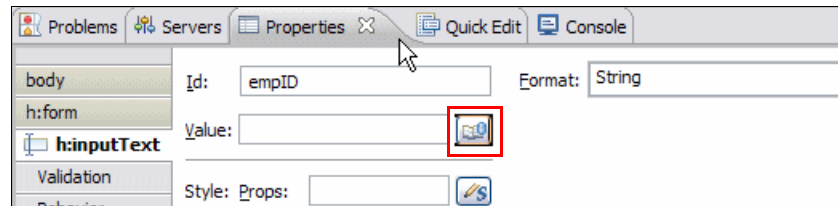


Figure 8-32 Input properties

- a. At the resulting Select Data Page Object pop-up, select **findEmployeeManagedBean** → **empId** and click OK.

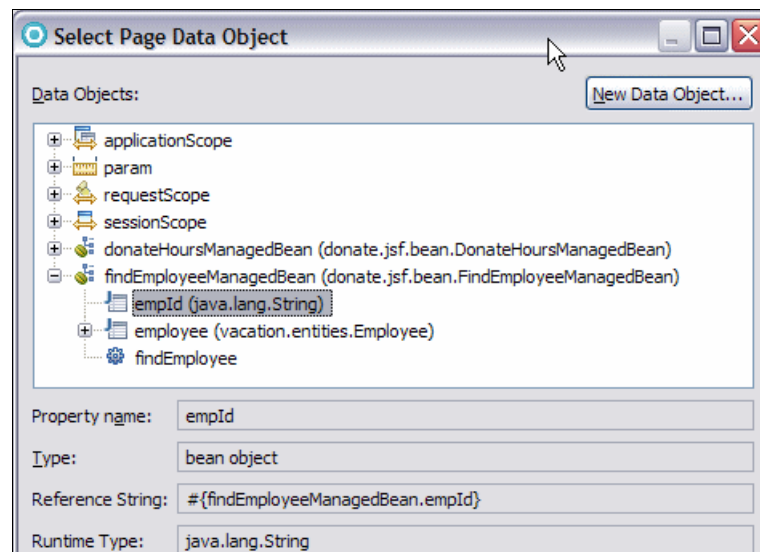


Figure 8-33 Select the data page object

- b. Back at the Properties view, note that the Value field now contains **`#{findEmployeeManagedBean.empId}`**.



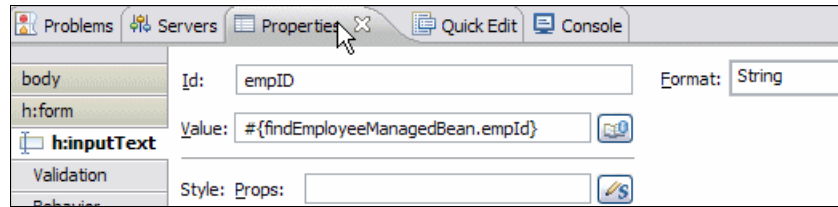


Figure 8-34 Input properties

**Behind the scenes:** This instructs JSF to place the user input into a property named `empId` in the `FindEmployee` managed bean.

```
<h:inputText styleClass="inputText" id="empID" maxLength="8"
size="10" required="true"
value="#{findEmployeeManagedBean.empId}">
```

You can use also context assist within the editor source pane to help with the binding:

- ▶ Insert `#{}`  between the double quotes for the value parameter.
- ▶ Place the cursor between the braces and press **Ctrl+Space**.
- ▶ Select the managed bean (`findEmployeeManagedBean`) from the list.
- ▶ Type a period and press **Ctrl+Space** again and select the property (`empId`).

2. In the editor design pane, select the command (Submit) button in the JSP,
  - a. In the Properties view, scroll to the right, next to the Action or Outcome field enter `#{findEmployeeManagedBean.findEmployee}`.

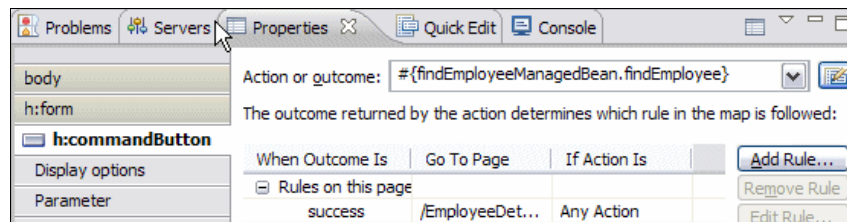


Figure 8-35 Command button properties

### Behind the scenes:

The Action or outcome field has a browse button similar to that used for the input field, but it will only function if you have generated page code beans.

3. Example 8-6 on page 326 shows the source of the completed FindEmployee page.
  - a. Save and close FindEmployee.jsp.

#### *Example 8-6 FindEmployee.jsp code*

```
!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><%@page
    language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<html>
<head>
<title>Donate - Search</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" title="Style"
    href="theme/stylesheet.css">
</head>
<f:view>
    <body>
        <h:form styleClass="form" id="form1">
            <h:outputText styleClass="outputText" id="text1"
value="EmployeeID:"></h:outputText>
            <h:inputText styleClass="inputText" id="empId"
                value="#{findEmployeeManagedBean.empId}"></h:inputText>
            <h:commandButton type="submit" value="Submit" label="Submit"
                styleClass="commandButton" id="submit"

action="#{findEmployeeManagedBean.findEmployee}"></h:commandButton>
        </h:form>
    </body>
</f:view>
</html>
```


### Behind the scenes:

Notice the `<%@ taglib>` tags that define the JSF libraries. The body of the page is a JSF `<view>` with an embedded `<form>` that submits data to the server.

### 8.6.3 Implement the EmployeeDetails page

The EmployeeDetails JSP contains a table with the employee data, and an input field to donate hours of vacation time.

To implement the EmployeeDetails page, follow these steps:

1. Open **EmployeeDetails.jsp** (in DonateJSFWeb/WebContent) with the Page Designer editor.
2. Set the title tag to <title>Donate - Employee</title>.
3. Drag an **Output** component from the JSF HTML Palette into the page, and in the Properties view **h:outputText** tab:
  - a. Set Value to **Employee data**.
  - b. Next to **Style: Props:** click the **Browse** icon [  ].
  - c. At the resulting Set Style Properties pop-up:
    - i. On the left select the Font tab, and on the right set Size to **24**.
    - ii. On the left select the Font → Font Styles tab, and on the right set Weight to **bold**.
    - iii. Click **OK**.

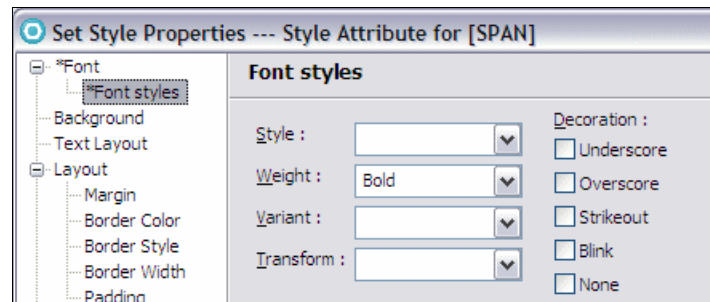
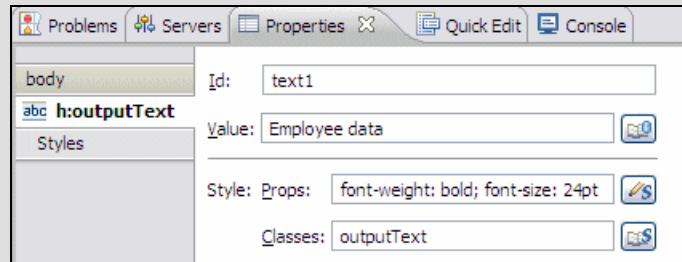


Figure 8-36 Set the style properties for the output component

**Behind the scenes:** the changes from this pop-up should be reflected across views:

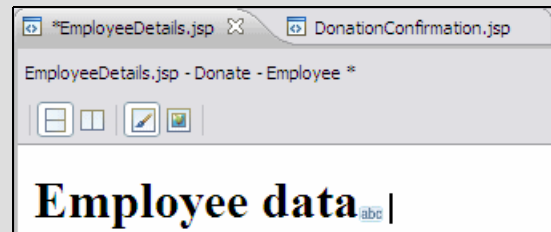
- In the Properties view:



- In the editor source pane:

```
<h:outputText styleClass="outputText" id="text1"
value="Employee data"
style="font-weight: bold; font-size:
24pt"></h:outputText>
```

- In the editor design pane:



4. Enable the Panel Grid component on the Palette:
  - a. Select anywhere on the Palette view, right-click and select **Customize**.
  - b. At the resulting Customize Palette pop-up:
    - i. On the right, expand to and select **Standard Faces Component** → **Panel Grid**.
    - ii. On the right, clear the **Hide** check box, and Click **OK**.

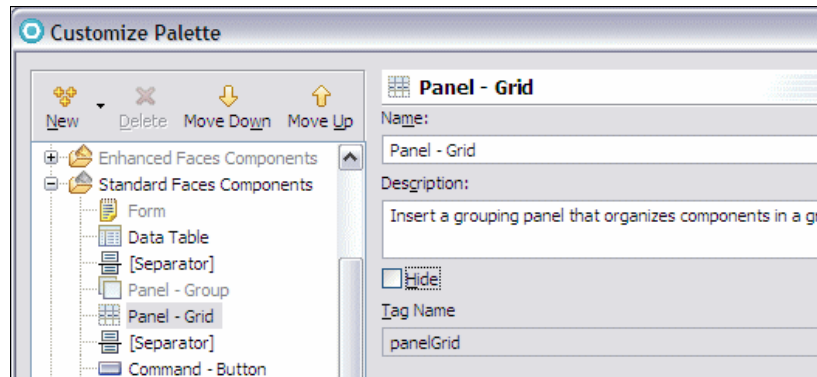


Figure 8-37 Customize the palette

**Behind the scenes:** Rational Application Developer reduces the complexity for the novice developer by showing only the most common Faces elements in the palette. In this case, you require access to one of the components that Rational Application Developer had hidden.

5. Drag a **Panel Grid** component into the page below the Employee Data output field.
  - a. In the Properties view **h:panelGrid** tab, set Columns to **2**. Note that the panel grid now contains two cells at the frame.

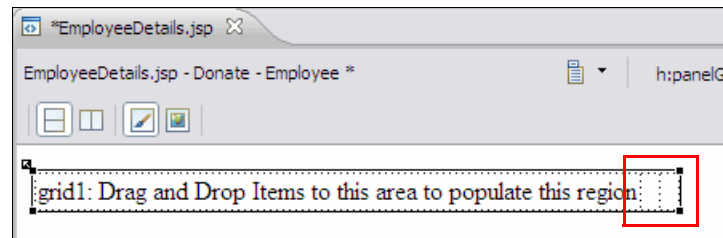


Figure 8-38 Panel grid component properties

- b. Drag an Output component into the page and drop into the first cell at the end of the panel grid.
  - c. Drag and drop 13 additional Output components onto the first Output component. This should create seven output rows within the panel grid.

```
<h:panelGrid styleClass="panelGrid" id="grid1" columns="2">
  <h:outputText styleClass="outputText" id="text14"></h:outputText>
```

```

        <h:outputText styleClass="outputText" id="text13"></h:outputText>
        ...
        ...
        <h:outputText styleClass="outputText" id="text2"></h:outputText>
        <h:outputText styleClass="outputText" id="text1"></h:outputText>
    </h:panelGrid>

```

6. For each row, select the **Output Text** component in the first cell and in the Properties view set the Value as follows (with an extra space pad character at the end of each):

- Employee ID:
- First name:
- Middle name:
- Last name:
- Vacation hours:
- Salary:
- Donate hours:

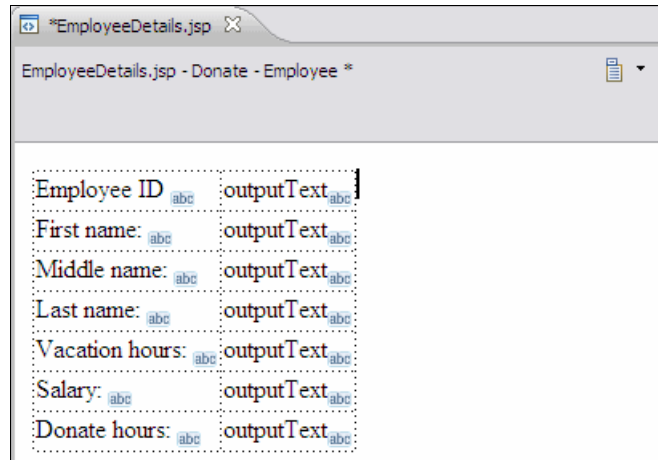


Figure 8-39 Output component text settings

7. For rows 1 - 6, select the **Output Text** component in the second cell and in the Properties view:
  - a. Next to Value click the **Browse** button
  - b. At the resulting Select Data Page Object pop-up, select the corresponding field (for example, **findEmployeeManagedBean** → **employee** → **employeeID**) and click **OK**.

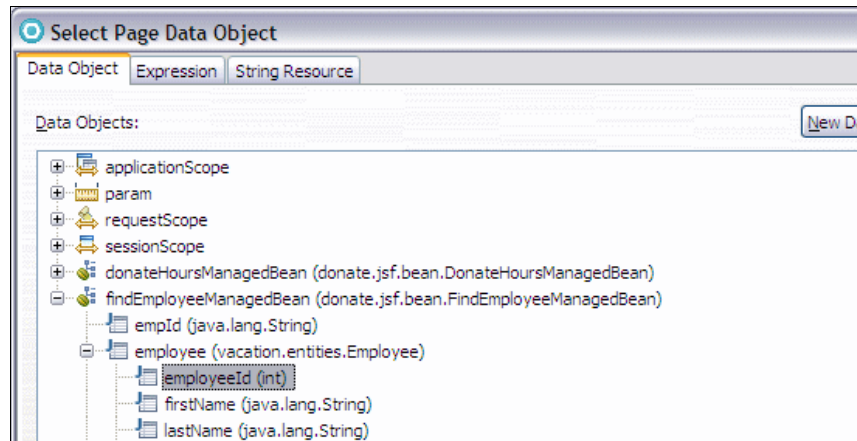


Figure 8-40 Output component text settings

- c. Do not change the second cell in row 7 at this time.
- d. The editor design pane should show the following after you complete the update for the first six rows:



Figure 8-41 Output component text settings

8. Delete the remaining unchanged second outputText component on the seventh row, and replace with an Input component from the Palette view (drag and drop onto the end of the Donate hours outputText component).
  - a. In the Properties view h:inputText tab, set **Value** to **#{donateHoursManagedBean.hoursDonated}**.
  - b. In the editor source tab, set the maxlength to 3 and size to 5.

**Behind the scenes:** the updated input stanza should be similar to:

```
<h:inputText styleClass="inputText" id="text1" maxLength="3"
size="5"
value="#{donateHoursManagedBean.hoursDonated}"></h:inputText>
```

The panelGrid will also be surround by a <form> tag.

9. Drag and drop a **Command Button** component after the input field.

a. In the Properties view h:commandButton tab:

- i. Ensure that Type is set to **Submit**.
- ii. Set Action or outcome to **#{donateHoursManagedBean.donateHours}**.

**Behind the scenes:** the updated command button stanza should be similar to:

```
<h:commandButton type="submit" value="Submit" label="Submit"
styleClass="commandButton" id="button1"
action="#{donateHoursManagedBean.donateHours}">
</h:commandButton>
```

10. Review the completed JSP design view. Example 8-7 on page 333 shows the source of the EmployeeDetails page.

- Save and close EmployeeDetails.jsp.

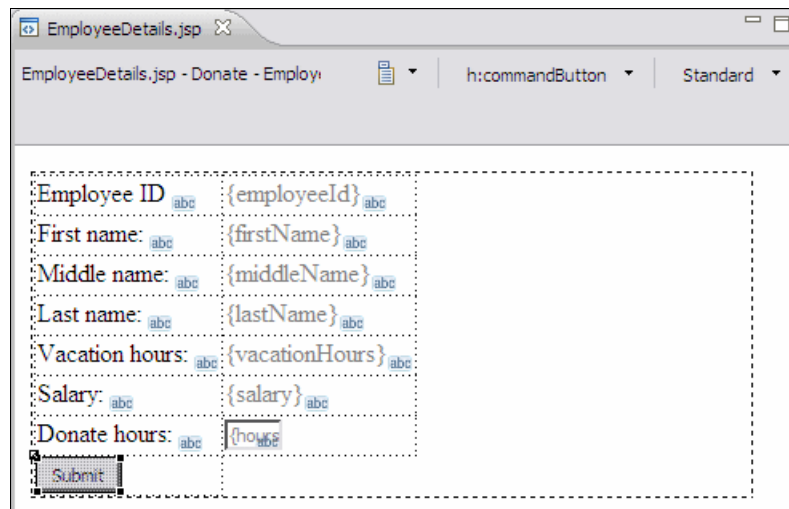


Figure 8-42 EmployeeDetails JSP



*Example 8-7 EmployeeDetails.jsp code*

---

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><%@page
    language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<html>
<head>
<title>Donate - Employee</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" title="Style"
    href="theme/stylesheet.css">
</head>
<f:view>
    <body>
        <h:form styleClass="form" id="form1">
            <h:panelGrid id="panelGrid" id="grid1" columns="2">
                <h:outputText styleClass="outputText" id="text14"
                    value="Employee ID "></h:outputText>
                <h:outputText styleClass="outputText" id="text13"

value="#{findEmployeeManagedBean.employee.employeeId}"></h:outputText>
                <h:outputText styleClass="outputText" id="text12"
                    value="First name: "></h:outputText>
                <h:outputText styleClass="outputText" id="text11"

value="#{findEmployeeManagedBean.employee.firstName}"></h:outputText>
                <h:outputText styleClass="outputText" id="text10"
                    value="Middle name: "></h:outputText>
                <h:outputText styleClass="outputText" id="text9"

value="#{findEmployeeManagedBean.employee.middleName}"></h:outputText>
                <h:outputText styleClass="outputText" id="text8" value="Last name:
"></h:outputText>
                <h:outputText styleClass="outputText" id="text7"

value="#{findEmployeeManagedBean.employee.lastName}"></h:outputText>
                <h:outputText styleClass="outputText" id="text6"
                    value="Vacation hours: "></h:outputText>
                <h:outputText styleClass="outputText" id="text5"

value="#{findEmployeeManagedBean.employee.vacationHours}"></h:outputText>
                <h:outputText styleClass="outputText" id="text4" value="Salary:
"></h:outputText>
                <h:outputText styleClass="outputText" id="text3"

value="#{findEmployeeManagedBean.employee.salary}"></h:outputText>
                <h:outputText styleClass="outputText" id="text2"
```

```

        value="Donate hours: "></h:outputText>
        <h:inputText styleClass="inputText" maxLength="3" size="5"
id="text1"

value="#{donateHoursManagedBean.hoursDonated}"></h:inputText>
        <h:commandButton type="submit" value="Submit"
label="Submit"
        styleClass="commandButton" id="button1"
action="#{donateHoursManagedBean.donateHours}"></h:commandButton>
    </h:panelGrid>
</h:form>
</body>
</f:view>
</html>

```

---

## 8.6.4 Implement the DonationConfirmation page

The DonationConfirmation JSP contains the result of donating vacation hours.

To implement the DonationConfirmation page, follow these steps:

1. Open **DonationConfirmation.jsp** (in DonateJSFWeb/WebContent) with the Page Designer editor.
2. Set the title tag to `<title>Donate - Result</title>`.
3. Drag an **Output Text** component into the page.
  - In the Properties view h:outputText tab, set Value to **`#{donateHoursManagedBean.resultMessage}`**.

**Behind the scenes:** resultMessage contains the result string from the donateVacation invocation.

4. With the cursor at the end of the output field, press **Enter** to insert a new line.
5. Drag an **Output Text** component into the page (after the new line). In the Properties view h:outputText tab, set the Value to **Thank you for donating your vacation hours.**
6. With the cursor at the end of the output field, press **Enter** to insert a new line.
7. Drag an **Output Text** component into the page (after the new line). In the Properties view h:outputText tab, set the Value to **You have the following amount of vacation hours left:** (with a space).
8. Drag an **Output Text** component into the page (after the [previous Output Text component]). In the Properties view h:outputText tab, set the Value to **`T#{donateHoursManagedBean.employee.vacationHours}`**.

**Behind the scenes:** vacationHours contains the amount of vacation that the employee has left after the donation.

9. Review the completed JSP design view. Example 8-8 shows the source of the DonationConfirmation page.
  - a. Save and close DonationConfirmation.jsp.

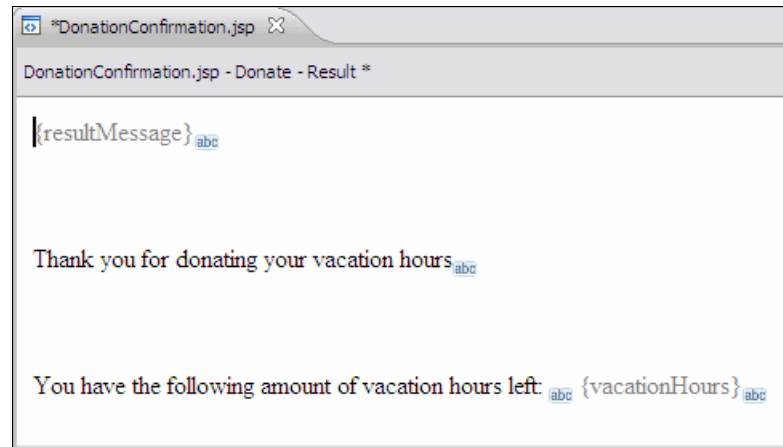


Figure 8-43 DonationConfirmation.jsp

**Example 8-8** DonationConfirmation.jsp code

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><%@page
    language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<html>
<head>
<title>Donate - Result</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" type="text/css" title="Style"
    href="theme/stylesheet.css">
</head>
<f:view>
    <body>
        <h:outputText styleClass="outputText" id="text1"
value="#{donateHoursManagedBean.resultMessage}"></h:outputText>
        <p></p>
        <h:outputText styleClass="outputText" id="text2" value="Thank you for
donating your vacation hours"></h:outputText>
        <p></p>
```

```

        <h:outputText styleClass="outputText" id="text3" value="You have
        the following amount of vacation hours left: "></h:outputText>
        <h:outputText styleClass="outputText" id="text4"
        value="#{donateHoursManagedBean.employee.vacationHours}"></h:outputText>
        <p></p>
        </body>
    </f:view>
</html>

```

---





## 8.6.5 Recapitulation

In the preceding steps:

- ▶ We defined the three pages of the JSF application.
- ▶ We wired the navigation path throughout the application using the `faces-config.xml`.
- ▶ We also provided the invocation of the business functionality behind each page, by providing two managed beans with data and action code.
- ▶ We used JSF value binding `#{...}` to bind JSF input and output fields to properties of the managed beans.
- ▶ We used JSF action binding to bind Submit buttons to methods in the managed beans.

## 8.7 Test the JSF Web application

Implementation-wise, we have completed the functionality for pages in this application. Now, it is time to see our implementation in action:

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to:
    -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`
  - and execute:
    -  `startNetworkServer.bat`
    -  `startNetworkServer.sh`
  - If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.

- If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.

#### Behind the scenes:

Recall that you started both of these in earlier chapters. They should still be running unless you stopped them or restarted the system.

- ▶ 4.2, “Start the Derby Network Server” on page 87.
- ▶ 3.6, “Start the ExperienceJEE Test Server” on page 78.

2. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
3. Select **FindEmployee.jsp** (in DonateJSFWeb/WebContent), right-click and select **Run as** → **Run on server**. At the Run On Server pop-up:
  - a. Select **localhost** → **ExperienceJEE Test Server**.
  - b. Enable Always use this server when running this project.
  - c. Click **Finish**.

**Behind the scenes:** you can also change this setting from the Servers tab on the DonateWeb properties pop-up.

4. The initial page is displayed (note that it takes a while the first time to compile the JSP). Enter an employee number and click **Submit**.

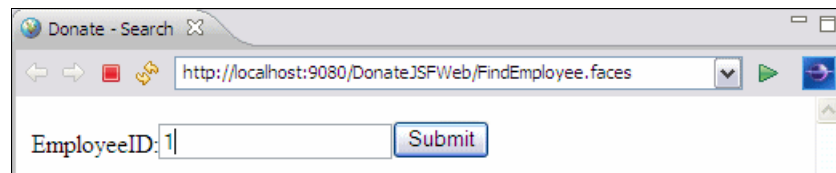


Figure 8-44 FindEmployee JSP initial page

### Off course?

If no action occurs after you click submit, you neglected to set the command action in 8.6.2, “Bind FindEmployee to a managed bean” on page 323

```
<h:commandButton type="submit" value="Submit" label="Submit"
    styleClass="commandButton" id="submit"
    action="#{findEmployeeManagedBean.findEmployee}">
</h:commandButton>
```

5. The employee is retrieved and the employee data is displayed on the FindEmployee page. Enter a number of hours (1) to donate, and click **Submit**.

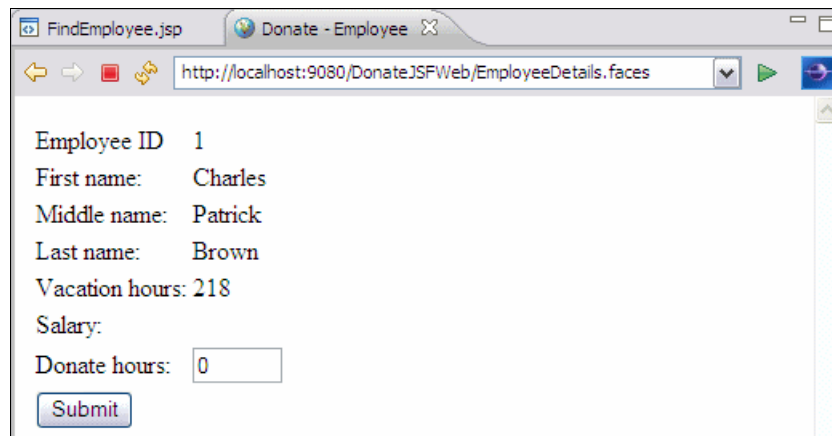


Figure 8-45 Enter input and submit

6. The donation is executed and the result is displayed on the DonationConfirmation page.

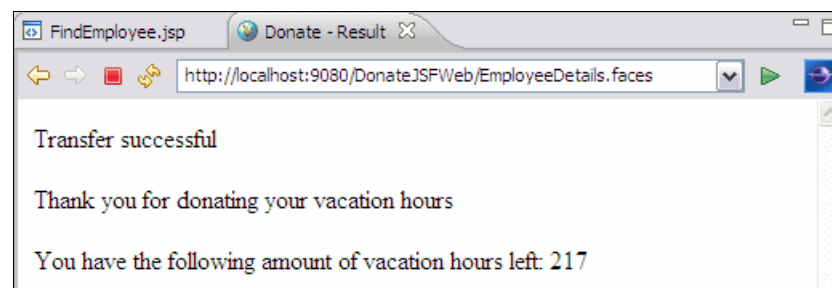
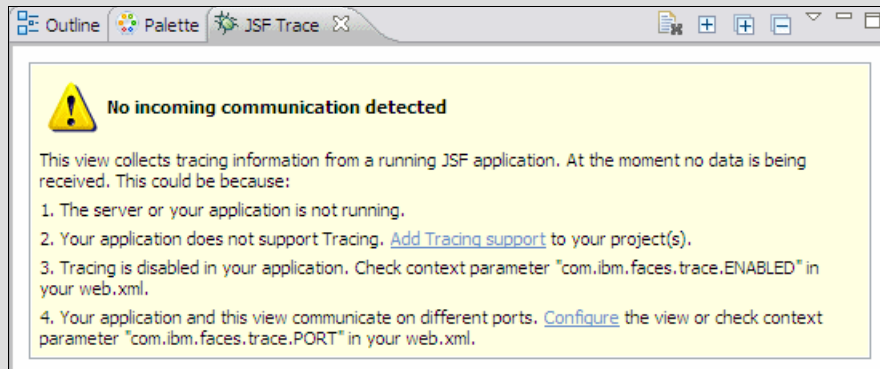


Figure 8-46 Results

### Behind the scenes:

You should have seen the JSF Trace view open in the right-hand pane after executing FindEmployee.jsp:



JSF Trace enables you to view trace messages from your JSF requests. This is not wire-level monitoring (like the TCP/IP monitor described in 11.7.1, “Monitor the SOAP traffic using TCP/IP Monitor” on page 437) but instead this instruments the actual JSF library to allow to you view what is occurring inside your application.

JSF Trace is intended for testing and development and not for production use due to the performance implications if in depth tracing.

Refer to the Rational Application Developer Information Center *What is JSF trace?* topic for further details:

<http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.etools.jsf.doc/topics/cjsftrace.html>

## 8.8 Optional extensions to DonateJSFWeb

The preceding sections provide a basic JSF application and all that is needed to complete Chapter 10, “Implement core security” on page 361.

The following sections show how to extend DonateJSFWeb to provide additional capabilities that are useful in a more robust application.

## 8.8.1 Error handling

The current JSF application does not provide any error handling, for example, when an employee is not found. If you enter an invalid employee number in FindEmployee, the current application shows no evident error to the user -- the browser window stays on the current page with no indication that anything occurred -- successful or unsuccessful.

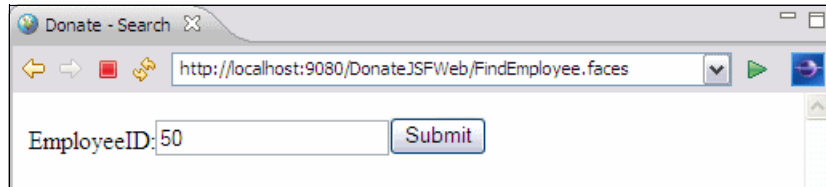


Figure 8-47 No error shown for invalid ID

The ExperienceJEE Test Server log in the Console view does show the error, but this information is not passed on to the user:

```
...
0000003c SystemErr      R java.lang.Exception: Employee with id 50 not found
0000003c SystemErr      R at
...
```

### JSF error message components

JSF provides two components to display errors:

- ▶ Message: Displays messages that are attached to another component, for example, an input field.
- ▶ Messages: Displays all error messages in a list or table.

### Adding an error message to find an employee


To add an error message for an invalid employee ID, perform these steps:

1. Provide a field for error messages in the FindEmployee page:
  - a. Open **FindEmployee.jsp** (in DonateJSFWeb/WebContent)
  - b. Drag and drop a **Display Errors** component (Standard Faces Components).



Figure 8-48 Add a Display Errors component



- c. In the Properties View h:messages tab, set Layout to **Display as List**.
- d. In the Properties View h:messages tab, next to **Style: Props:** click the **Browse** icon [  ]
  - i. At the New Style pop-up, set Color to **red** and click **OK**.
- e. Save and close FindEmployee.jsp

**Behind the scenes:** the above steps should result in the following source tag in FindEmployee.jsp:

```
<h:messages styleClass="messages" id="messages1" layout="table"
style="color: red"></h:messages>
```

2. Insert the error message into the messages field by the action code in the managed bean:
  - a. Open **FindEmployeeManagedBean.java** (in DonateJSFWeb/Java Resources: src/donate.jsf.bean).
  - b. Add the lines in bold to the findEmployee method.

```
...
...
    catch(Exception e) {
        e.printStackTrace();
        FacesContext ctx = FacesContext.getCurrentInstance();
        ctx.addMessage("", new FacesMessage(e.getLocalizedMessage()));
    }
    return ""; // stay on the same page
}
```

- c. Organize imports (**Ctrl+Shift+O**) to resolve:
  - import javax.faces.application.FacesMessage;
  - import javax.faces.context.FacesContext;
- d. Save and close FindEmployeeManagedBean.java.

### Behind the scenes:

This code creates a Faces message when an employee is not found, and adds it to the Faces context. JSF displays all messages added to the Faces context in the Messages component.

You can also provide the id of a component in the addMessage method to display the message in a specific Message component. The empty id ("" ) directs the message to the Messages component.

3. Test the FindEmployee error message:
  - a. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
  - b. Select **FindEmployee.jsp** (in DonateJSFWeb/WebContent), right-click and select **Run as** → **Run on server**.
  - c. At the resulting browser window, enter an invalid employee number (such as 50) and click **Submit**.
    - i. Verify that an error message is displayed for an invalid employee ID:

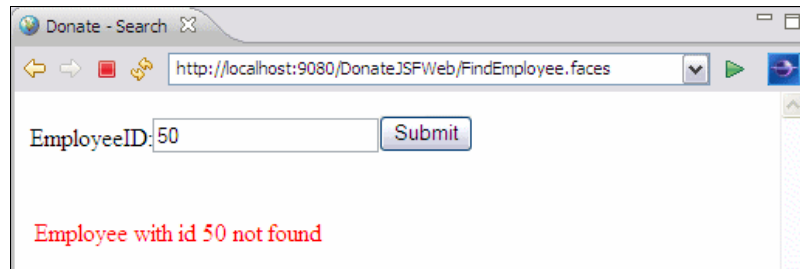



Figure 8-49 Error message

## 8.8.2 Validation

JSF provides elaborate validation of data, for example, when typing values of the wrong data type. In addition, we can add custom validation by value range, for example, for the employee ID:

1. Open **FindEmployee.jsp** (in DonateJSFWeb/WebContent):
2. In the resulting editor, select the input field in the design pane.
3. In the Properties view Validation tab:
  - a. Select the **Click to create/edit custom validation code** icon[  ].

- b. At the resulting Quick edit view, select the input area ("Insert a code snippet or write....")
- i. At the resulting Create Page Code File pop-up, click **OK**.

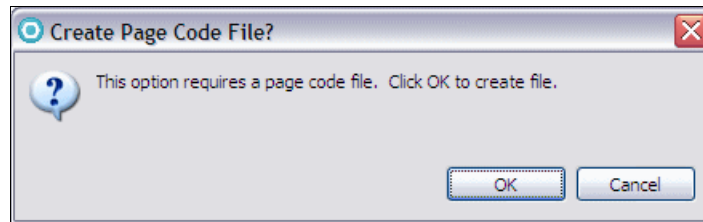


Figure 8-50 Create Page Code File pop-up

**Behind the scenes:** custom validators require a backing page code managed bean.

- c. Back at the Quick edit view, click **Open in Java Editor**.

**Behind the scenes:** the code you will add requires import statements, and they cannot be defined from within the Quick edit view.

- d. In the resulting FindEmployee.java editor:
  - i. Insert the **F08.8 EmployeeID Validator** snippet from the Snippets view between the comments in the handleEmpIdValidate1 method (Example 8-9)

Example 8-9 F08.8 EmployeeID Validator and surrounding code

```
public void handleEmpIdValidate1(FacesContext facescontext,
    UIComponent component, Object object)
    throws javax.faces.validator.ValidatorException {
    // Type Java code to handle validate here
    int empId = 0;
    try {
        empId = Integer.parseInt(object.toString());
    } catch (Exception e) {
        throw new ValidatorException(new FacesMessage(
            "Value is not of the correct type."));
    }
    if (empId < 1 || empId > 99) {
        throw new ValidatorException(new FacesMessage("Value " + empId
            + " not between the expected range of 1 and 99"));
    }
}
```

```

    }
    // void validate(FacesContext facescontext, UIComponent component,
    // Object object) throws ValidatorException
}

```

---

ii. Organize imports (**Ctrl+Shift+O**) to resolve:

```

import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;

```

e. Save and close FindEmployee.java.

4. Switch back to the FindEmployee.jsp source view and note the validator attribute that has been added to the inputText tag:

```

<h:inputText styleClass="inputText" id="empId"
    value="#{findEmployeeManagedBean.empId}" required="false"
    validator="#{pc_FindEmployee.handleEmpIdValidate1}">
</h:inputText>

```

– Save and close FindEmployee.jsp.

5. Test the FindEmployee validation:

- a. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
- b. Select **FindEmployee.jsp** (in DonateJSFWeb/WebContent), right-click and select **Run as** → **Run on server**.
- c. At the resulting browser window, enter an out of bound employee number (such as 200) and click **Submit**. Observe that you receive a JSF error message: Value 200 not between the expected range of 1 and 99
  - i. Type an alphabetic employee ID and you receive the JSF error message: Value is not of the correct type.

## 8.9 Explore! JSF

For more resources on JSF, consult these IBM Redbooks publications:

- ▶ *Rational Application Developer V7 Programming Guide*, SG24-7501, Chapter 14, *Develop Web applications using JSF and SDO*. Note that instead of Service Data Objects (SDO) we use JPA entities to access the database.
- ▶ *Rational Application Developer V7.5 Programming Guide*, SG24-7672, Chapter 16, *Develop Web applications using JSF*, where we also use JSF with JPA entities.



## Create the application client

In this chapter, we create simple non-graphical client applications that can interact with the server side DonateBean session EJB:

- ▶ Java EE application client that provides *thick client* access to the session EJB.
- ▶ Standalone client that provides a *thinner* client access to the session EJB.

## 9.1 Learn!

Figure 9-1 shows a simple non-graphical client applications that can interact with the server side DonateBean session EJB.

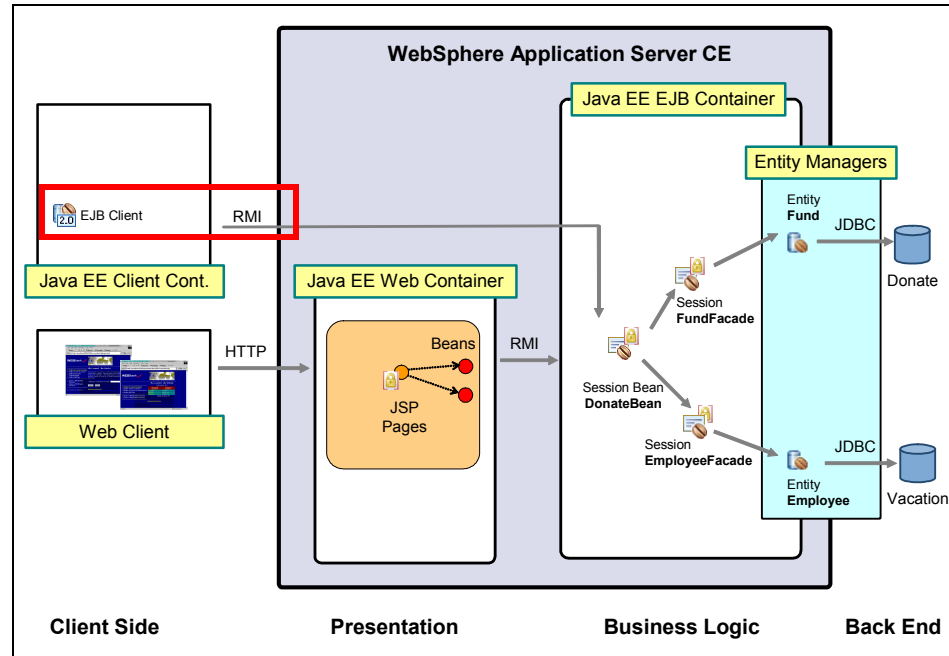


Figure 9-1 Donate application: Java EE application client

The traditional Web application created in the previous chapter renders all content on the client's system using a Web browser. This is called a *thin* client, which means that no unique application code is required. However, this type of client is not appropriate in certain situations, such as these:

- ▶ The client system requires sub-second response time. For example, customer service applications often require very fast response times to ensure that customer calls are processed as quickly as possible. It is often unacceptable to wait—for even one second—for a Web browser to send the request to the server to display an alternate page.
- ▶ The client system requires high graphics resolution or complex graphics display capabilities that cannot be accommodated by the browser.
- ▶ The client system requires asynchronous notification from the server side, using messaging (JMS), for example. A standard browser is driven by the user, which means that nothing happens until the user presses Enter or until the browser page is automatically refreshed using a timer.

For these situations, Java EE provides a *thick client* called the Java EE application client that consists of a runtime infrastructure that must be installed on the client system along with the actual application. The user can invoke this application, which then interacts with the Java EE server-side environment using resources such as JNDI, EJBs, JDBC, and JMS.

The one drawback of the application client is that it is a thick client: It requires the installation (or download) of a Java EE runtime environment that consists of Java libraries and configuration files.

As a result, many developers prefer to create a smaller informal runtime environment that consists only of the required JAR files and configuration files that can be easily downloaded and executed on demand. This is sometimes called a standalone client.

The Java EE specification also defines the Applet container which supports the execution of Java programs in the context of a browser session. Although this is formally defined in Java EE, in reality applets have no direct access to server-side Java EE services and thus can be treated as standalone clients.

The examples used in this chapter are simple, non-graphical clients. However, most real-world application clients are implemented using a graphical display capability such as Swing, Abstract Window Toolkit (AWT), or Standard Widget toolkit (SWT).

There are no specific Learn resources for this chapter.

## 9.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed these steps:

- ▶ Defined the two databases and the matching database pools
- ▶ Created the JPA entities and the three session beans
- ▶ Created and run the JUnit test cases
- ▶ Optionally created and tested the Web application

## 9.3 Create the application client project

In this section you create the application client project that will contain the main Java class and any other required class path definitions/artifacts:

1. In the Java EE Perspective Enterprise Explorer, select **DonateEAR**, right-click and select **New** → **Application Client Project**.
2. At the New Application Client Project/Application Client module pop-up:
  - a. Set these values:
    - Project name to **DonateClient**.
    - Application Client module version to **5.0**.
    - Configuration to **Default Configuration for WebSphere Application Server v7.0**.
  - b. Under EAR membership, select **Add project to an EAR**, and select **DonateEAR**.
  - c. Click **Finish**.

### Behind the scenes:

The application client project would be in a different enterprise application in most real world examples. We keep it simple here and use one enterprise application for the complete example.

## 9.4 Configure the application client project

We must add DonateEJB to the manifest for DonateClient. This is required so that the Main class has access to the donateToFund method that we created in the DonateBean session EJB (in Chapter 7, “Create the Donate session bean for the business logic” on page 261):

1. In the Enterprise Explorer, select **DonateClient**, right-click and select **Properties**.
2. At the resulting Properties for DonateClient pop-up:
  - a. On the left, select **Java EE Module Dependencies**.
  - b. On the right, enable **DonateEJB.jar**.
  - c. Click **OK** to save the changes.

### Behind the scenes:

The DonateEJB.jar file is added to the MANIFEST.MF file, and makes the classes in the EJB project available to the client project.



## 9.5 Code the main method

As with any standalone Java application, a main method is required to start the application. The Main class was created in the default package. Use the following instructions to code the main method:

1. In the Enterprise Explorer, open **Main.java** (in DonateClient/appClientModule/ (default package)). Note that the class currently contains the following main method:

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
}
```

2. In the Main editor, and insert the following lines, before the main method:

```
@EJB  
static DonateBeanRemote aDonate;
```

3. In the main method replace the // TODO line with a blank line, and insert the **F09.1 App Client Main** snippet from the Snippets view (Example 9-1).

*Example 9-1 F09.1 App Client Main and surrounding code*

---

```
public static void main(String[] args) {  
  
    if (args.length == 3) {  
        int employeeId = Integer.parseInt((args[0]));  
        String fundName = args[1];  
        int hours = (new Integer(args[2])).intValue();  
  
        System.out.println("Employee Number = " + employeeId);  
        System.out.println("Fund           = " + fundName);  
        System.out.println("Donation Amount = " + hours);  
  
        System.out.println("Donation result = " +  
            aDonate.donateToFund(employeeId, fundName, hours));  
    } else {  
        System.out.println("Program requires 3 command line arguments.");  
    }  
}
```

---

4. Organize the imports (**Ctrl+Shift+O**) to resolve:

```
import javax.ejb.EJB;  
import donate.ejb.interfaces.DonateBeanRemote;
```

5. Save and close Main.java.

### Behind the scenes:

Behind the scenes these activities are happening:

- Initialize the session bean instance through annotation:

```
@EJB
static DonateBeanRemote aDonate;
```

Note how much easier this is with EJB 3.0, instead of having to locate the home of the session bean through JNDI.

- Verify that the appropriate arguments were passed to the main method:

```
if (args.length == 3) {
    int employeeId = Integer.parseInt((args[0]));
    String fundName = args[1];
    int hours = (new Integer(args[2])).intValue();

    System.out.println("Employee Number = " + employeeId);
    System.out.println("Fund           = " + fundName);
    System.out.println("Donation Amount = " + hours);
    ...
} else {
    System.out.println("Program requires 3 command line arguments.");
}
```

- Call the `donateToFund` method and print out the results:

```
System.out.println("Donation result = " +
    aDonate.donateToFund(employeeId, fundName, hours));
```

## 9.6 Test the application client



Java EE application clients require a configured runtime environment called the Client container. This is similar to the Web container used for Servlets and JSPs and the EJB container used for EJBs. With WebSphere Application Server the following applies:

- The EJB and Web containers are provided by installing the Application Server component.
- The Client container is provided by installing the WebSphere Application Client, typically on a machine that does NOT have the Application Server.




The Application Server component also installs the Client container for the rare situation where you want to run an Application Client on the same system as your server (where the client is in one JVM and the server is in another JVM).

This development environment scenario, using the Rational Application Developer workbench and the embedded WebSphere Application Server test environment, is such a rare situation: running an Application Client on the same system as the Application Server.

1. Ensure that the test environment is running:

- If the Derby Network Server is not running, open a command window, change directory to
  -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
  -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`

and execute:

-  `startNetworkServer.bat`
-  `startNetworkServer.sh`
- If the ExperienceJEE Test Server is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
  - If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.

2. In the Enterprise Explorer, select **DonateClient**, right-click and select **Run As** → **Run Configurations...**

3. At the Create, Manage, and run Configurations pop-up:

- a. On the left, under Configurations, select **WebSphere v7.0 Application Client**, right-click and select **New**.

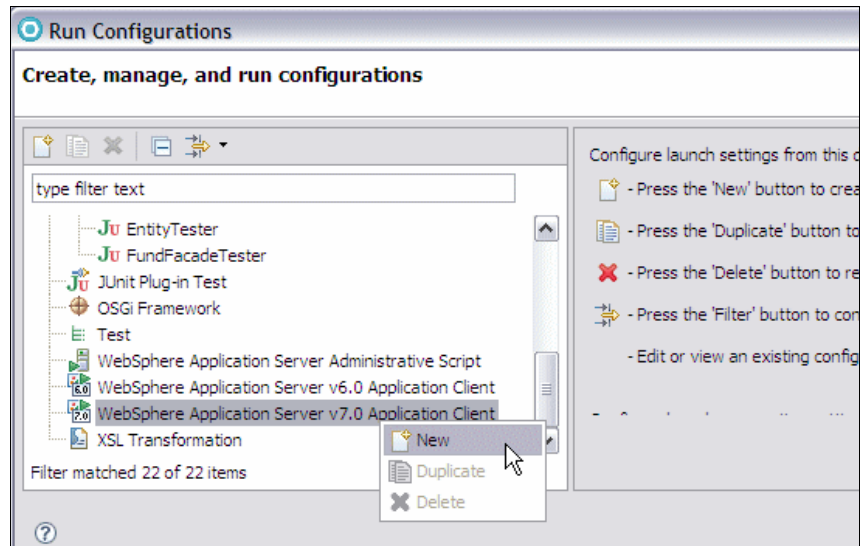


Figure 9-2 Select the Run configuration

- b. On the resulting right pane set Name to **DonateClient**.
- c. On the right pane Application tab:
  - WebSphere Runtime: **WebSphere Application Server V7**
  - Enterprise application: **DonateEAR**
  - Application client module: **DonateClient**
- d. Select **Enable application client to connect to a server**.
- e. Select **Use specific server**, and then select **ExperienceJEE Test Server**
- f. On the right pane Arguments tab:
  - Append **1 DonationFund 1** to the program argument string, making sure that you leave a space between it and the existing arguments. The resulting program arguments string should be similar to the following:

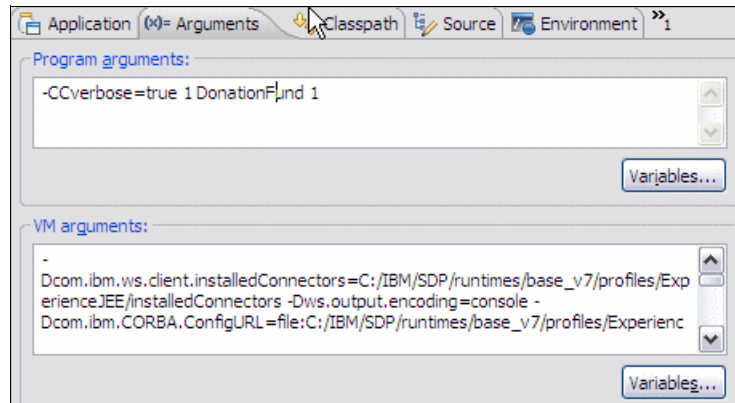


Figure 9-3 Program arguments

- g. Click **Apply** to save the configuration (the pop-up remains open).
  - h. Click **Run** to execute the configuration (and the pop-up closes).
4. At the Login at the Target Server pop-up, click **Cancel**.

**Behind the scenes:** the application client launch configuration automatically queries for JAAS credentials, but DonateClient does not require any credentials at this point.

Note that will change once you enable application security at which point you will need to provide a userid and password. Refer to 10.7, “Update the Java EE application client for security” on page 401 for further details.


**Off course?** Sometimes this pop-up may be obscured, so if you do not see it, and the **Console** view appears to be stopped after these messages, try minimizing some other windows until you see this.

```
...
...
WSCL0911I: Component initialized successfully.
WSCL0901I: Component initialization completed successfully.
```

5. Switch to the Console view to observe the execution results

```
WSCL0911I: Component initialized successfully.
WSCL0901I: Component initialization completed successfully.
WSCL0035I: Initialization of the Java EE Application Client Environment has
completed.
WSCL0014I: Invoking the Application Client class Main
```

```
Employee Number = 1
Fund             = DonationFund
Donation Amount = 1
Donation result = Transfer successful:
```

6. You can rerun the application client by several methods:
  - a. From the workbench action bar, click on the run icon [] and select **DonateClient** from the pull-down list.
  - b. Select **Run** → **Run Configurations** and from the resulting pop-up select select **WebSphere Application Server v7.0 Application Client** → **DonateClient**, and click **Run**. Note that you can also change the program arguments using this method.
7. The application client can also be executed from a command line:
  - a. Export the DonateEAR enterprise application
    - i. In the Enterprise Explorer, select **DonateEAR**, right-click and select **Export** → **JAR file**.
    - ii. At the Export/EAR Export pop-up, set Destination to **C:\7827code\DonateEAR.ear** and click **Finish**.
  - b. From a command window, switch to the profile bin directory and execute the launch client command:

```
C:
CD C:\IBM\SDP\runtimes\base_v7\profiles\ExperienceJEE\bin
launchclient C:\DonateEAR.ear 1 DonationFund 1
```
  - c. At the Login at Target Server pop-up, click **Cancel**. The command execution completes:

```
WSCL0035I: Initialization of the Java EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class Main
Employee Number = 1
Fund             = DonationFund
Donation Amount = 1
Donation result = Transfer successful
```

**Off Course?** If the application client is empty, switch to the ExperienceJEE Test Server output and review for any errors.

If you see the following error:

```
[12/30/09 10:21:29:734 EST] 00000038 SSLHandshakeE E SSLHandshake :
Unable to initialize SSL connection. Unauthorized access was
denied or security settings have expired. Exception is
javax.net.ssl.SSLException: Unrecognized SSL message, plaintext
connection?
    at com.ibm.jsse2.b.a( )
    at com.ibm.jsse2.pc.a( )
    at com.ibm.jsse2.pc.unwrap( )
    at javax.net.ssl.SSLEngine.unwrap( )
    ...
```

the problem is that the run configuration used the wrong paths when configuring the VM arguments. The application client does not have the proper SSL certificates and the access fails.

This problem has occurred with certain levels of Rational Application Developer when you execute a run configuration against a non-default WebSphere Application server profile. The run configurations are configured with the default profile path, and as a result they access an incorrect path and fail as described above.

For example, if you used a Rational Application Developer configuration that contained a default WebSphere Application Server V7.0 profile that used a wasprofile1 directory, then you might see this error when connecting to the ExperienceJEE Test Server/profile.

The work around is to manually edit the VM arguments on the run configuration Arguments tab and change all occurrences of the profile directory (in this case change wasprofile1 to ExperienceJEE):

```
-Dcom.ibm.ws.client.installedConnectors=C:/IBM/SDP/runtimes/bas
e_v7/profiles/was70profile1/installedConnectors
-Dws.output.encoding=console
-Dcom.ibm.CORBA.ConfigURL=file:C:/IBM/SDP/runtimes/base_v7/prof
iles/was70profile1/properties/sas.client.props
-Dcom.ibm.SOAP.ConfigURL=file:C:/IBM/SDP/runtimes/base_v7/profi
les/was70profile1/properties/soap.client.props
...
...
```

## 9.7 Create and test a standalone client

In this section we create a standalone Java client that accesses the DonateBean session bean. Instead of starting with a new project, we can reuse the DonateUnitTester project and add a main method to the DonateBeanTester class:

1. In the Enterprise Explorer, open **DonateBeanTester.java** (in DonateUnitTester/src/donate.test).
  - a. Insert the **F09.2 Standalone Client Main** snippet into the class before the closing bracket (Example 9-2).
  - b. Save and close DonateBeanTester.java.

*Example 9-2 F09.2 Standalone client main method*

---

```
public static void main(String[] args) throws Exception {
    DonateBeanTester.init();
    if (args.length == 3) {
        int employeeId = Integer.parseInt((args[0]));
        String fundName = args[1];
        int hours = (new Integer(args[2])).intValue();

        System.out.println("Employee Number = " + employeeId);
        System.out.println("Fund           = " + fundName);
        System.out.println("Donation Amount = " + hours);

        System.out.println("Donation result = " +
            donateBean.donateToFund(employeeId, fundName, hours));
    } else {
        System.out.println("Program requires 3 command line arguments.");
    }
    DonateBeanTester.tearDown();
}
```

---



### Behind the scenes:

Behind the scenes these activities are occurring:

- ▶ Initialize the session bean instance by calling the JUnit initialization method (which looks up the EJB directly from the WebSphere Application Server JNDI).

```
DonateBeanTester.init();
```

- ▶ Verify that the appropriate arguments were passed to the main method:

- ▶ Call the `donateToFund` method and print out the results:

```
System.out.println("Donation result = " +  
    donateBean.donateToFund(employeeId, fundName, hours));
```

2. In the Enterprise Explorer, select **DonateBeanTester.java** (in `DonateUnitTester/src/donate.test`), right-click and select **Run As → Run Configurations**.
3. At the Create, manage, and run configurations pop-up:
  - a. In the left pane, select **Java Application**, right-click and select **New**.
  - b. In the right pane, select the **Arguments** tab and set Program arguments to **1 DonationFund 1**
  - c. Click **Apply** to save the changes.
  - d. Click **Run** to execute the standalone client.

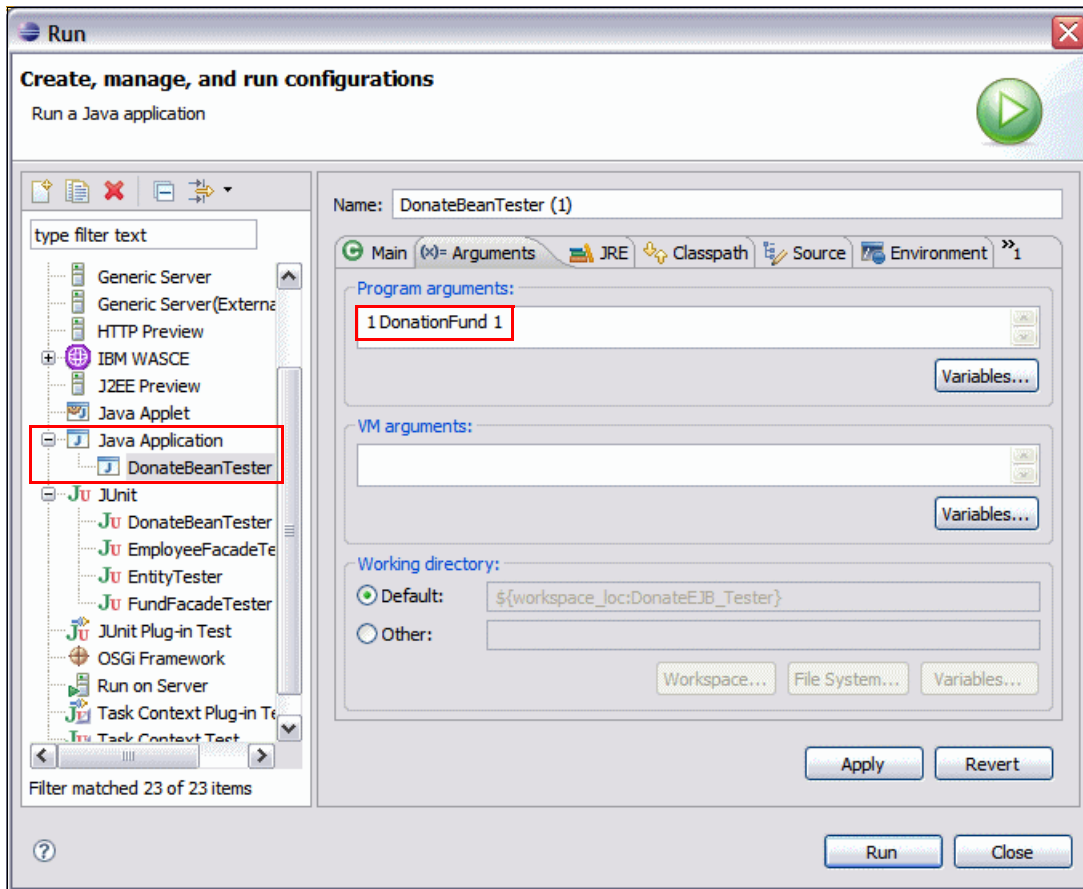


Figure 9-4 Define and run the configuration

4. The Java application results are displayed in the Console view:

```

DonateBeanTester: Start
Dec 30, 2009 10:34:25 AM null null
WARNING: WSVR0072W
Dec 30, 2009 10:34:26 AM null null
WARNING: WSVR0072W
Dec 30, 2009 10:34:26 AM null null
WARNING: WSVR0072W
Dec 30, 2009 10:34:26 AM null null
WARNING: WSVR0072W
Employee Number = 1
Fund           = DonationFund
Donation Amount = 1
Donation result = Transfer successful
TearDown: number of funds=0

```

DonateBeanTester: EndDonateBeanTester: End

**Behind the scenes:** the WSVR0072W messages can be ignored. They are somewhat common when interacting with WebSphere Application server. Code updates to reduce the occurrence of these is slated for WebSphere Application Server V7.0 fixpack 7 (7.0.0.7).

5. You can rerun the client by selecting **Run** → **Run Configurations**, selecting **Java Application** → **DonateBeanTester (1)**, changing the arguments, clicking **Apply**, and clicking **Run**.

## 9.8 Explore!

As extra credit, you can run the standalone client from a Java command line:

1. Export the DonateUnitTester project to a JAR file:
  - a. In the Enterprise Explorer, select **DonateUnitTester**, right-click and select **Export**.
  - b. At the Export/Select pop-up, select **Java** → **JAR file** and click **Next**.
  - c. At the JAR Export/Jar File Specification pop-up, set Jar file to C:\7827code\appclient\DonateUnitTester.jar and click Finish.
2. Export the enhanced DonateEJB.jar (under DonateUnitTester) as a file:
  - a. In the Enterprise Explorer, select **DonateEJB.JAR** (under DonateUnitTester), right-click and select **Export**.
  - b. At the Export/Select pop-up, select **General** → **File System** and click **Next**.
  - c. At the Export/Jar File system pop-up, set To directory to **C:\7827code\appclient** and click **Finish**.
3. Export the DonateJPAEmployee and DonateJPAFund projects each to a JAR file by selecting **Export** → **EJB JAR file**, set the names to C:\7827code\appclient\DonateJPAEmployee.jar, and C:\7827code\appclient\DonateJPAFund.jar.
4. Run the following command in an operating system prompt:

```
C:\IBM\SDP\runtimes\base_v7\java\bin\java -cp
DonateUnitTester.jar;DonateEJB.jar;DonateJPAEmployee.jar;DonateJPAFund.jar;
C:\IBM\SDP\runtimes\base_v7
\runtimes\com.ibm.ws.ejb.thinclient_7.0.0.jar donate.test.DonateBeanTester
1 DonationFund
```

**Behind the scenes:** A sample command file to run the standalone client is provided in C:\78327code\appclient\run.bat.

Note that we specified the Java executable packaged with the WebSphere Application Server test environment. If you left the java command unqualified (no directory/path) and if the default Java executable was a lower level, then the execution could fail with errors similar to the following:

```
The java class could not be loaded.  
java.lang.UnsupportedClassVersionError:  
(donate/test/DonateBeanTester) bad major version at offset=6
```



## Implement core security

In this chapter, we secure Web and Java EE application client access to the application.

We define a security realm in the server, and implement groups and users. Then we define declarative security to restrict access to the session EJB and a JSF Web page. Using programmatic security, we hide sensitive salary information in the session EJB and in the JSF Web application. Finally, we update the Java EE application client to force a security login.

## 10.1 Learn!

Figure 10-1 shows how we secure the application.

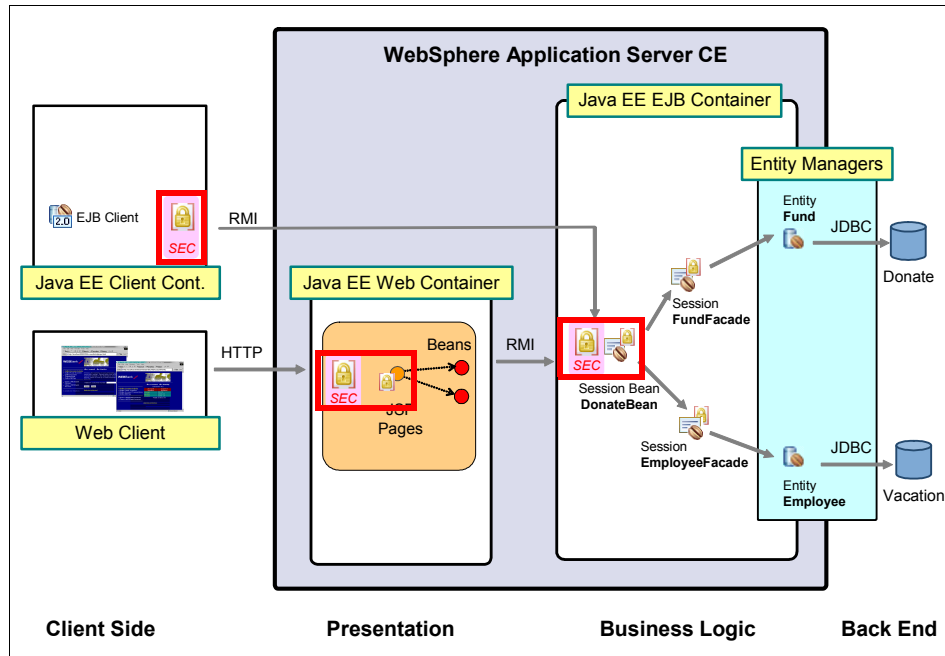


Figure 10-1 Donate application: Java EE application security

Java EE contains a multi-level security architecture for applications. This chapter focus on the following aspects:

- ▶ User registry, which defines the valid users and groups within a Java EE environment
- ▶ Application role based security, which restricts access to Java EE resources based on roles, which are then mapped to users and groups
- ▶ Administrative security

Following are other security aspects that are briefly discussed in the Explore section:

- ▶ Single sign-on
- ▶ Java 2 Security
- ▶ Java Authentication and Authorization Service (JAAS)
- ▶ Java Authorization Contract for Containers (JACC)

Additional Learn resources include these:

- ▶ The Java EE 5 Tutorial: Chapter 28: Introduction to Security in the Java EE Platform:  
<http://java.sun.com/javaee/5/docs/tutorial/doc/bnbwj.html>
- ▶ The Java EE 5 Tutorial: Chapter 29: Securing Java EE Applications:  
<http://java.sun.com/javaee/5/docs/tutorial/doc/bnbyk.html>
- ▶ The Java EE 5 Tutorial: Chapter 30: Securing Web Applications:  
<http://java.sun.com/javaee/5/docs/tutorial/doc/bncas.html>

### 10.1.1 User registry

Any Java EE application server that utilizes application role-based security or administrative security (which is discussed in the subsequent sections) requires a mechanism for authenticating users and groups.

The Java EE mechanism for this type of security is a user registry, and WebSphere supports the following to store/access user IDs and passwords:

- ▶ Database
- ▶ Lightweight Directory Access Protocol (LDAP)
- ▶ Properties file
- ▶ Certificates (X.509 certificates)

Access to a secured Java EE resource results in a request against the user registry to validate the presented credentials (user ID/password).

Note that the user registry by itself does not enforce any security policies or constraints. It simply validates the supplied user, group, and password information that is presented to it by the Java EE authentication mechanisms.

### 10.1.2 Application role-based security

Application role-based security allows the developer to secure access to both Web and EJB resources based on roles.

Often the developer has no knowledge of the users and groups that will exist at runtime, so how does the developer refer to the set of users and groups that should have access to a specific artifact?

The answer is through *roles*. Java EE allows the developer to define an arbitrary set of roles and to restrict access to artifacts based on these roles. Later, the developer (or an administrator) can map these roles to actual users and groups.

Java EE has two forms of role-based security:

- ▶ **Declarative** security, where the security constraints limiting access are defined in the deployment descriptors, and the Java EE runtime is the mechanism that manages access to the Java EE artifact. Consider this as coarse-grained security, where the user will see all or none of the employee data.
- ▶ **Programmatic** security, where the user code queries if a user is in a role, and then executes different statements depending on whether the user is in the role or not. Consider this as fine-grained security, where depending on their role, the user might only see a subset of the employee data.

### 10.1.3 Administrative security

All Java EE application servers should have secure access to their administrative interfaces (such as the WebSphere Application Server Administrative Console GUI and wsadmin command line interface), and these are usually implemented using the Java EE application role based security model.

In the case of WebSphere Application Server, this implementation allows the following predefined roles: Administrator, Operator, Configurator, Monitor, iscadmins, and adminsecuritymanager.

Starting with WebSphere Application Server V7.0, the administrative security can be enabled independently from the standard application security. This allows the administrator to run the overall applications without security but still manage the actual server with security.

## 10.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A., “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed these steps:

- ▶ Defined the two databases and the matching database pools
- ▶ Created the JPA entities and the three session beans
- ▶ Created and run the JUnit test cases
- ▶ Created the Web applications
- ▶ Created the application client



## 10.3 Preliminary security setup

In this section you create the users and groups for use in demonstrating application role based security:

Table 10-1

Group	User members
DUSERS	DCBROWN, DNARMSTRONG
DMANAGERS	DNARMSTRONG
<no group>	DGADAMS

WebSphere allows you to define the users in the file-based registry by manually editing the file, through the Admin Console, or by using the **wsadmin** command.

The following sections add the users and groups using the **wsadmin** command.

### 10.3.1 Create and configure the Jython project and script

Jython is the recommended scripting language for WebSphere administration. In this section you use the workbench to create a Jython project, and then create the Jython script that adds the users and groups.

1. In the Java EE perspective Enterprise Explorer select anywhere on the view, right-click and select **New** → **Other...**
  - a. At the New/Select a wizard pop-up, select **Jython** → **Jython Project**, and click **Next**.
  - b. At the New Jython project/Create a Jython project pop-up, set the Project Name to **DonateJython**, and click **Finish**.
2. In the Enterprise Explorer, select **DonateJython**, right-click and select **New** → **Other...**
  - a. At the New/Select a wizard pop-up, select **Jython** → **Jython Script File**, and click **Next**.
  - b. At the New Jython Script File/Create a Jython script file pop-up, set the File name to **AddGroupsUsers.py**, and click **Finish**.
3. Switch to the open **AddGroupsUsers.py** editor (**DonateJython/AddGroupsUsers.py**):
  - a. Select the cursor on the last blank line, and insert the **F10.1 AddGroupsUsers** snippet from the Snippets view.

### Example 10-1 F10.1: AddGroupsUsers

---

```
AdminTask.createGroup('-cn DUSERS')
AdminTask.createGroup('-cn DMANAGERS')
AdminTask.createUser('-uid DCBROWN -password xxxxxxxx
-confirmPassword xxxxxxxx -cn DCBROWN -sn DCBROWN')
AdminTask.createUser('-uid DNARMSTRONG -password xxxxxxxx
-confirmPassword xxxxxxxx -cn DNARMSTRONG -sn DNARMSTRONG')
AdminTask.createUser('-uid DGADAMS -password xxxxxxxx
-confirmPassword xxxxxxxx -cn DGADAMS -sn DADAMS')
AdminTask.addMemberToGroup('-memberUniqueName
uid=DCBROWN,o=defaultWIMFileBasedRealm -groupUniqueName
cn=DUSERS,o=defaultWIMFileBasedRealm')
AdminTask.addMemberToGroup('-memberUniqueName
uid=DNARMSTRONG,o=defaultWIMFileBasedRealm -groupUniqueName
cn=DUSERS,o=defaultWIMFileBasedRealm')
AdminTask.addMemberToGroup('-memberUniqueName
uid=DNARMSTRONG,o=defaultWIMFileBasedRealm -groupUniqueName
cn=DMANAGERS,o=defaultWIMFileBasedRealm')
```

---

- b. Change all six occurrences of the string xxxxxxxx to match the password you want to use for these users. For example, change the following line from:

```
AdminTask.createUser('-uid DCBROWN -password xxxxxxxx
-confirmPassword xxxxxxxx -cn DCBROWN -sn DCBROWN')
```

to:

```
AdminTask.createUser('-uid DCBROWN -password mypassword
-confirmPassword mypassword -cn DCBROWN -sn DCBROWN')
```

4. Save and close AddGroupsUsers.py.

## 10.3.2 Run the Jython script

In this section you execute the Jython script to add the users and groups.

1. In the Enterprise Explorer select **AddGroupsUsers.py** (in DonateJython), right-click and select **Run As** → **Administrative Script**.
2. At the resulting **Edit Configuration / Edit configuration and launch** pop-up:
  - a. Ensure that you are in the **Script** tab.
  - b. Set the Scripting runtime environment to **WebSphere Application Server v7.0**.
  - c. Set the Profile name to **ExperienceJEE**.
  - d. Under **Security** select **Specify**.

- i. Enter the javaeeadmin user ID and password defined in section 3.4, “Create the ExperienceJEE server profile” on page 62.
- e. Click **Apply**.
- f. Click **Run**.

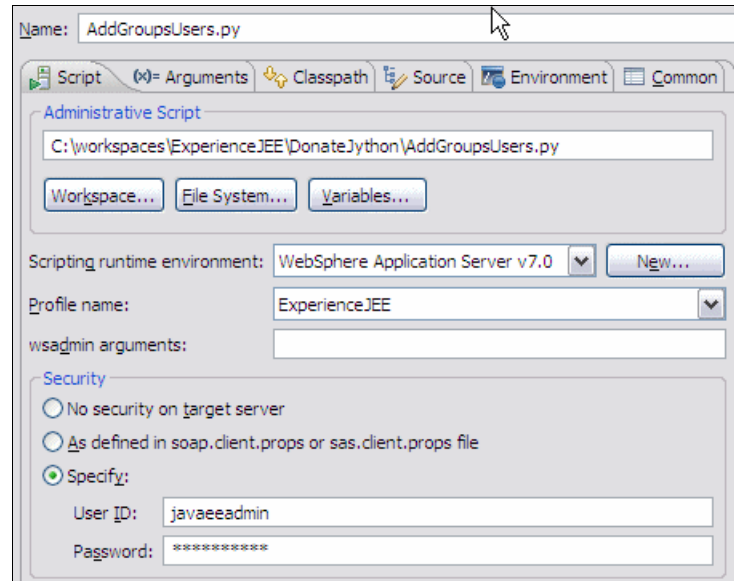


Figure 10-2 Edit the script runtime configuration

3. In the Console view, monitor the output of the AddGroupsUsers.py script. If the script works successfully, you will see a single message indicating that the script connected to the server.

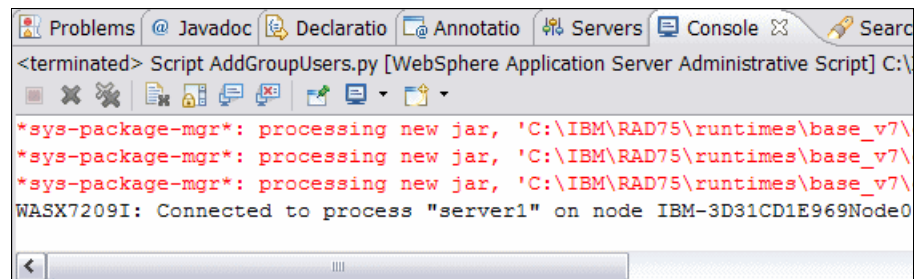


Figure 10-3 Output of the script

Note that the first time this script runs additional messages appear indicating that various jar files are being processed:

```
*sys-package-mgr*: processing new jar,  
'C:\IBM\SDP\runtimes\base_v7\optionalLibraries\jython\jython.jar'  
*sys-package-mgr*: processing new jar,  
'C:\IBM\SDP\runtimes\base_v7\lib\startup.jar'  
*sys-package-mgr*: processing new jar, '...  
...
```

## Behind the scenes:

For the file-based repository, group (and user) entries are stored in fileRegistry.xml, found in the cell directory of the WebSphere profile (for example, in C:\IBM\SDP\runtimes\base\_v7\profiles\ExperienceJEE\config\cells\testwinNode01Cell).

The typical format for the group entry is as follows:

```
<wim:entities xsi:type="wim:Group">
  <wim:identifier
    externalId="40dfbd86-54ea-49f0-8b93-cc2758ea4f1b"
    externalName="cn=DMANAGERS,o=defaultWIMFileBasedRealm"
    uniqueId="40dfbd86-54ea-49f0-8b93-cc2758ea4f1b"
    uniqueName="cn=DMANAGERS,o=defaultWIMFileBasedRealm"/>
  <wim:parent>
    <wim:identifier uniqueName="o=defaultWIMFileBasedRealm"/>
  </wim:parent>
  <wim:createTimestamp>
    2006-07-07T07:25:29.765-05:00
  </wim:createTimestamp>
  <wim:cn>DMANAGERS</wim:cn>
</wim:entities>
```

You can also view and change these entries through the Admin Console. To see the current list of groups:

- ▶ In the Admin Console, on the left select **Users and Groups** → **Manage Groups**.
- ▶ On the right, in the **Manage Groups** panel, click **Search**. The current groups should appear as the following illustration shows.

**Search for Groups**

Search by Group name \* Search for \* \* Maximum results 100






2 groups matched the search criteria.

Select	Group name	Description	Unique Name
<input type="checkbox"/>	<a href="#">DMANAGERS</a>		cn=DMANAGERS,o=defaultWIMFileBasedRealm
<input type="checkbox"/>	<a href="#">DUSERS</a>		cn=DUSERS,o=defaultWIMFileBasedRealm

## 10.4 Enable Java EE application security

In this section you enable the Java EE application security. Note that you only have to enable the Java EE application security since the ExperienceJEE Test Server already has a configured user registry and already has Java EE administrative security enabled.

### 10.4.1 Enable Java EE application security

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to:
    -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`
  - and execute:
    -  `startNetworkServer.bat`
    -  `startNetworkServers.sh`
  - If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
    - If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.
2. Start the Admin Console using the same steps as described in 4.6.1, “Start the Administrative Console” on page 100. Recall that in the Admin Console login screen you use the javaeeadmin user ID and password that you defined in section 3.4, “Create the ExperienceJEE server profile” on page 62.
3. On the left select **Security** → **Global Security**.
4. On the right, in the Global Security panel:
  - a. Under Application security, select **Enable application security**.
  - b. Click Apply.

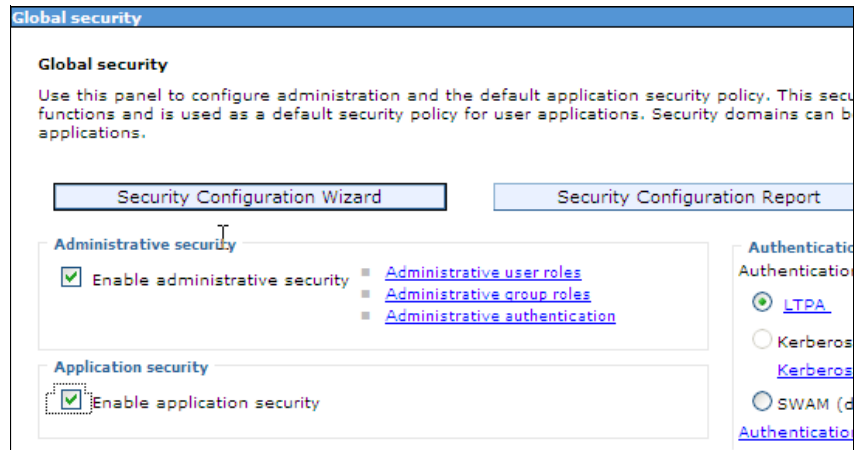


Figure 10-4 Enable application security

5. At the top of the panel, in the Messages box that states "Changes have been made to your local configuration...", click **Save**.
6. Logout, and close the Admin Console.

### Behind the scenes:

The message box also includes several warnings that can be ignored for this scenario:

The domain name for single signon is not defined. The Web browser defaults the domain name to the host name that runs the Web application. Single signon is restricted to the application server host name and does not work with other application server host names in the domain.

This book implements a single server single environment and therefore does not require single signon.

Further information about WebSphere Application Server single signon is available in the Information Center *Single sign-on for authentication* topic:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/csec\\_ssovo.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/csec_ssovo.html)

If the Restrict access to local resources option is not enabled, the Java virtual machine (JVM) system resources are not protected. For example, applications can read and write to files on file systems, listen to sockets, exit the Application Server process, and so on. However, by enabling the Restrict access to local resources option, applications might fail to run if the required permissions are not granted to the applications

The application development and test environment such as used in this book typically does not require the high level of security protections provided by the Java 2 security referenced by this comment.


In addition, enabling Java 2 security by default typically breaks many applications, and the developer goes through an extensive cycle of creating policies to grant the required permission for access to resources such as temporary files. Therefore, although this is a valid consideration, it is beyond the scope and intent of this book.

Further information about Java 2 security is available in the Information Center *Java 2 security* topic:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/csec\\_rsecmgr2.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/csec_rsecmgr2.html)



## 10.4.2 Restart the ExperienceJEE Test Server

1. In the Servers view, select **ExperienceJEE Test Server**, right-click and select **Restart**. You may alternatively click the (re)start icon [].

The server status changes to Stopping, then Starting, and finally back to Started when the startup is complete.

2. Switch to the Console view, and verify that the server started successfully. In particular look for the following message:

```
WSVR0001I: Server server1 open for e-business
```

## 10.5 Implement declarative security

Declarative security restricts access to URLs (servlets, JSPs, or even HTTP files or images served by the Web container) or EJBs. This is sometimes called coarse grained access because access is either allowed or disallowed:

- ▶ EJB access is controlled through annotations in the EJB class or through deployment descriptors in the EJB application.
- ▶ URL access is controlled through deployment descriptors in the Web application

The mapping of specific users and groups to the roles is defined in deployment descriptors, and this can be in the Web, EJB, or EAR deployment descriptor.

### 10.5.1 Restrict access to the Donate session EJB (declarative)

In this section, we use declarative security to restrict access to the DonateBean session EJB created in Chapter 7, “Create the Donate session bean for the business logic” on page 261:

1. In the Enterprise Explorer, open **DonateBean.java** (in DonateEJB/ejbModule/donate.ejb.impl).
2. In the resulting DonateBean editor:
  - a. Before the DonateBean class definition insert the following line:

```
@DeclareRoles( { "DMANAGERS_ROLE", "DUSERS_ROLE" })  
public class DonateBean implements DonateBeanInterface, ..... {
```

### Behind the scenes:

This annotation statement defines the Java EE roles that will be used for both declarative and programmatic security in this class. Recall that you defined the mapping of these roles to actual users and groups in 10.5.3, “Configure DonateEAR roles and role mapping” on page 380.

- b. Before both `donateToFund` methods insert a `@RolesAllowed` annotation:

```
@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
public void donateToFund(Employee employee, Fund fund, int hours) ...
.....

@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
public String donateToFund(int employeeId, String fundName, int hours) {
.....
}
```

### Behind the scenes:

These annotation statements restrict access to the `donateToFund` methods to users belonging to the `DMANAGERS_ROLE` and `DUSERS_ROLE` roles. Thus, user `DADAMS` will no longer be authorized to run these methods.

- c. Organize imports (**Ctrl+Shift+O**) to resolve:

```
import javax.annotation.security.DeclareRoles;
import javax.annotation.security.RolesAllowed;
```

- d. Save and close `DonateBean.java`.

**Deployment descriptor alternatives:** We inserted three annotation statements into the session EJB:

```
@DeclareRoles( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
@RolesAllowed( { "DMANAGERS_ROLE","DUSERS_ROLE" })
```

These statements can be replaced by the following statements in **ejb-jar.xml** (in `DonateEJB/ejbModule/META-INF`) immediately before the closing `</ejb-jar.xml>` tag:

```
<assembly-descriptor>
  <security-role>
    <role-name>DMANAGERS_ROLE</role-name>
  </security-role>
  <security-role>
    <role-name>DUSERS_ROLE</role-name>
  </security-role>
  <method-permission>
    <role-name>DUSERS_ROLE</role-name>
    <role-name>DMANAGERS_ROLE</role-name>
    <method>
      <ejb-name>DonateBean</ejb-name>
      <method-name>donateToFund</method-name>
    </method>
  </method-permission>
</assembly-descriptor>
```

The potential value of using the deployment descriptor instead of the annotations is that the security definitions can be changed at deployment time, without having access to the source code. Deployment descriptors always overwrite annotations.

## 10.5.2 Restrict access to Web pages via declarative security

In this section you implement Web declarative security to restrict access to the JSPs created in Chapter 8, “Create the Web front end” on page 277.

1. In the Enterprise Explorer, open **Security Editor** (in `DonateJSFWeb`).
2. At the Security Editor warning pop-up, click **OK**.

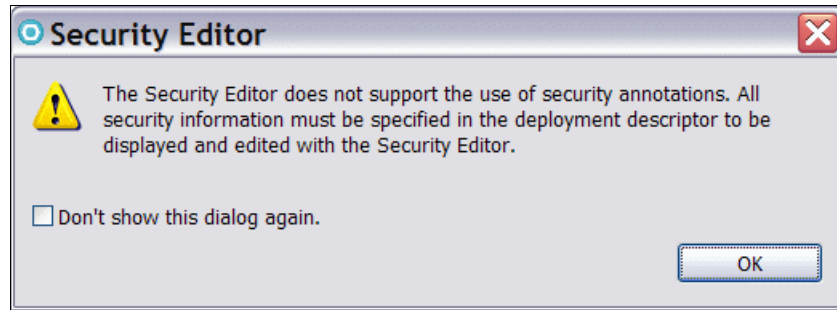


Figure 10-5 Security Editor warning pop-up

**Behind the scenes:** This generic warning has little meaning in the context of Web security since by definition all security statements must be in the deployment descriptor.

3. At the Create a Security Role warning pop-up (asking if you would like to create a security role), click **OK** because in fact that is the first security artifact that you need to create.
  - a. At the Add New Roles pop-up:
  - b. Set Role Name to **DUSERS\_ROLE** and click **Add**.
  - c. Set Role Name to **DMANAGERS\_ROLE** and click **Add**.
  - d. Set Role Name to **ALLAUTHENTICATED\_ROLE** and click **Add**.
  - e. Click **Finish**.

**Behind the scenes:** Three security role stanzas are added to the web.xml deployment descriptor (in DonateJSFWeb/WebContent/WEB-INF). Each is similar to:

```
<security-role>
  <description>
  </description>
  <role-name>DUSERS_ROLE</role-name>
</security-role>
```

4. In the Security Editor editor, assign the roles to the resources:
  - a. Under Resources, select **DonateJSFWeb** → **FindEmployee.jsp**, right-click and select **Assign Roles**:
    - i. At the Select Roles pop-up, select the **ALLAUTHENTICATED\_ROLE** and click **OK**.

- b. Under Resources, select **DonateJSFWeb** → **EmployeeDetails.jsp**, right-click and select **Assign Roles**:
      - i. At the Select Roles pop-up, select the **DUSERS\_ROLE** and **DMANAGERS\_ROLE** and click **OK**.
    - c. Under Resources, select **DonateJSFWeb** → **DonationConfirmation.jsp**, right-click and select **Assign Roles**:
      - i. At the Select Roles pop-up, select the **DUSERS\_ROLE** and **DMANAGERS\_ROLE** and click **OK**.
  5. Back in the Security Editor editor, select the authentication method.
    - a. In the lower right, click **Authentication**.
    - b. At the resulting User Authentication pop-up, set the Authentication Method to **Basic**.
    - c. Click **OK**.

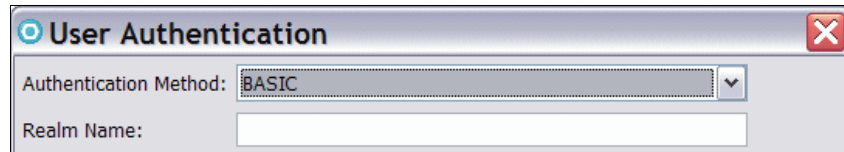


Figure 10-6 Select the authentication method

### Behind the scenes:

BASIC authentication instructs the application server to request the userid and password via a common login prompt, and the userid and password is encrypted using base64 encoding.

Other options include the following:

- ▶ **FORM**: Requests the authentication information through a customized logon page provided by the application developer.
- ▶ **Client\_CERT**: Uses digital certificates passed over an SSL connection.
- ▶ **Digest**: Similar to BASIC, but the password is transmitted using a custom encryption mechanism.

BASIC authentication is not very secure since the userid and password can easily be intercepted and decrypted. However, BASIC authentication combined with an SSL transport does protect the userid and password.

6. Back in the Security Editor editor, verify that the configuration is similar to Figure 10-7, and then save and close the Security Editor.

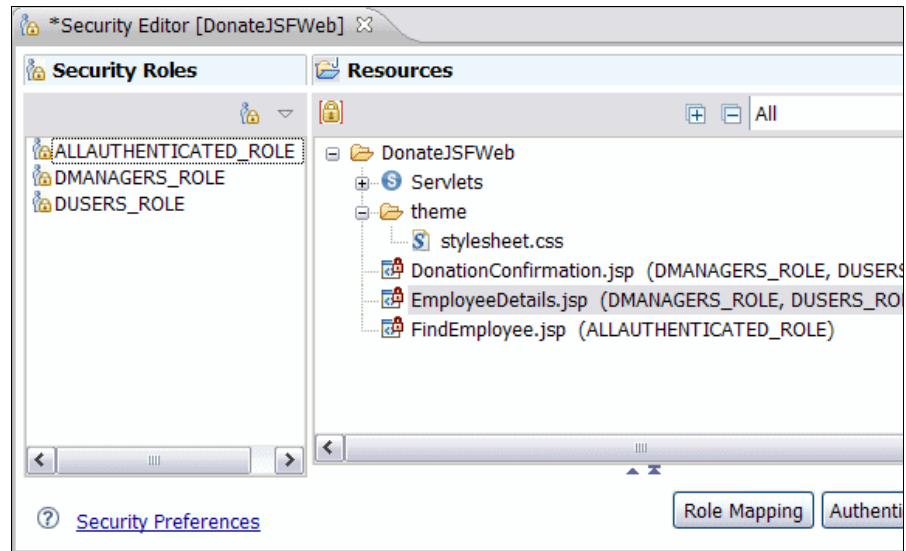


Figure 10-7 Security roles and resources configuration

## Behind the scenes:

The security editor inserts the following information into web.xml:

The security-constraint is the base element which contains:

- ▶ The <web-resource-collection element which defines the access privileges to a collection of resources. This element can specify which url patterns this security matches and during which http methods.
- ▶ The <auth-constraint> element which defines the role needed to access this web-resource-collection.

The security-role element declares the role-names used in all security-constraint elements. These names are mapped to the actual role names in the EAR binding configuration (which you will configure in the next section).

```
<security-constraint>
  <display-name>
    ALLAUTHENTICATED_ROLEConstraint</display-name>
  <web-resource-collection>

    <web-resource-name>ALLAUTHENTICATED_ROLECollection</web-resource-name>
      <url-pattern>/faces/FindEmployee.jsp</url-pattern>
      <url-pattern>/FindEmployee.faces</url-pattern>
      <url-pattern>/FindEmployee.jsp</url-pattern>
      <http-method>GET</http-method>
      <http-method>PUT</http-method>
      <http-method>HEAD</http-method>
      <http-method>TRACE</http-method>
      <http-method>POST</http-method>
      <http-method>DELETE</http-method>
      <http-method>OPTIONS</http-method>
    </web-resource-collection>
    <auth-constraint>
      <description>
        Auto generated Authorization Constraint</description>
      <role-name>ALLAUTHENTICATED_ROLE</role-name>
    </auth-constraint>
  </security-constraint>
<security-role>
  <description>
  </description>
  <role-name>DUSERS_ROLE</role-name>
</security-role>
```

### 10.5.3 Configure DonateEAR roles and role mapping

In this section, we configure the overall set of roles and mappings to users and groups in the DonateEAR enterprise application.

1. In the Enterprise Explorer, select DonateEAR, right-click and select **Java EE → Generate WebSphere Extensions Deployment Descriptor**.
  - a. Save and close the resulting `ibm-application-ext.xml` editor.
2. In the Enterprise Explorer, select DonateEAR, right-click and select **Java EE → Generate WebSphere Bindings Deployment Descriptor**.

**Off course?** If the editor fails to initialize and displays an exception, restart the workbench (from the workbench action bar select **File → Restart**).

Several readers have encountered situations where the wizard generation appears to have synchronization errors with the workbench, and restarting the workbench appears to resolve the problem.

3. In the resulting **ibm-application-bnd.xml editor** (in DonateEAR/META-INF), create the bindings for the ALLAUTHENTICATED\_ROLE:
  - a. Ensure that you are on the Design tab.
  - b. Under Overview, select Application Bindings, right-click and select **Add → Security Role**.

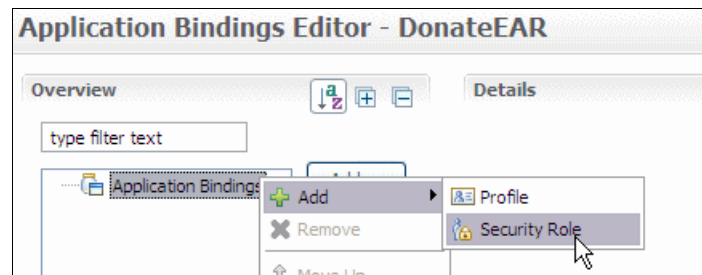


Figure 10-8 Add the security role to the application bindings

- c. Under Details set Name to **ALLAUTHENTICATED\_ROLE**.



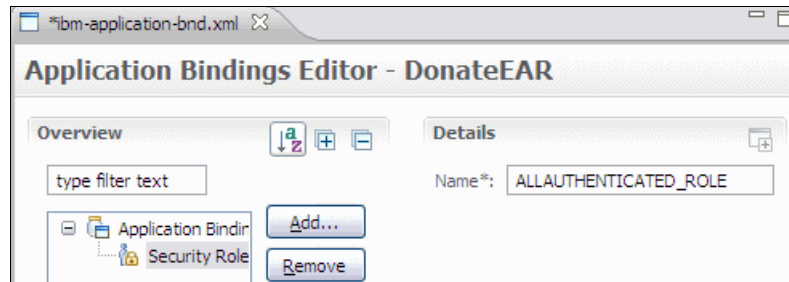


Figure 10-9 Define the role

- d. Under Overview select **Application Bindings** → **Security Role(ALLAUTHENTICATED\_ROLE)**, right-click and select **Add** → **Special Subject**.

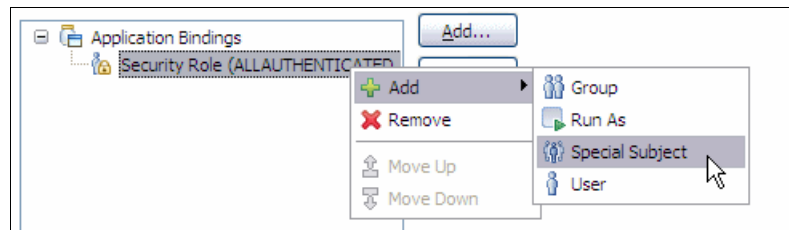


Figure 10-10 Assign a special subject to the role

- e. Under Details set Type to **All Authenticated Users** using the pull-down.

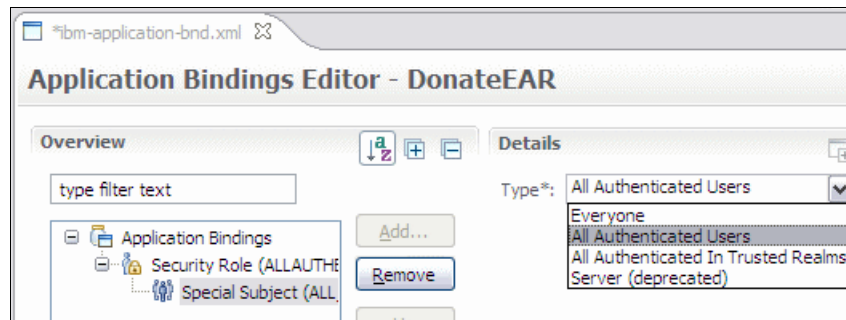


Figure 10-11 Special subject All Authenticated Users

4. Create the bindings for the DMANAGERS\_ROLE:
  - a. Under Overview, select Application Bindings, right-click and select **Add** → **Security Role**.

- b. Under Details set Name to **DMANAGERS\_ROLE**.
  - c. Under Overview select **Application Bindings** → **Security Role(DMANAGERS\_ROLE)** and click **Add** → **Group**.
  - d. Under Details set Name to **DMANAGERS**.
5. Create the bindings for the DUSERS\_ROLE:
  - a. Under Overview, select **Application Bindings**, right-click and select **Add** → **Security Role**.
  - b. Under Details set Name to **DUSERS\_ROLE**.
  - c. Under Overview select **Application Bindings** → **Security Role(DUSERS\_ROLE)** and click **Add** → **Group**.
  - d. Under Details set Name to **DUSERS**.
6. Verify that the Overview section is configured as shown in Figure 10-12 and save and close the `ibm-application-bnd.xml` Application Bindings editor.

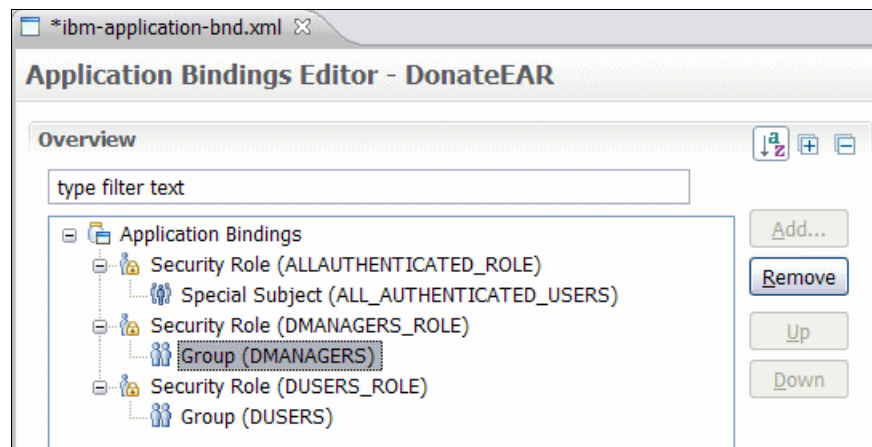


Figure 10-12 Application bindings for DonateEAR

### Behind the scenes:

This editor inserts the following information into the IBM binding file:

```
<security-role name="ALLAUTHENTICATED_ROLE">
  <special-subject type="ALL_AUTHENTICATED_USERS" />
</security-role>

<security-role name="DUSERS_ROLE">
  <group name="DUSERS" />
</security-role>
<security-role name="DMANAGERS_ROLE">
  <group name="DMANAGERS" />
</security-role>
```

## 10.5.4 Test the EJB declarative security using JUnit

We can test declarative security for the session bean using either JUnit or the Universal Test Client.

- ▶ JUnit can run with or without user credentials, thus allowing you to verify both denying and allowing unsuccessful access.
- ▶ The Universal Test Client only runs without user credential (from the perspective of Java EE role based security). Therefore, you can only verify for denying access.

You may optional skip either this section or the next (10.5.5, “Test the EJB declarative security using the UTC”). Only one is required to verify the EJB declarative security.

1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Verify that the access fails with the unaltered JUnit test, which accesses the DonateBean as unauthenticated.
  - a. In the Enterprise Explorer, select **DonateBeanTester.java** (in DonateUnitTester/src/donate.test), right-click and select **Run As → JUnit Test**.
  - b. In the JUnit view, verify that the donateToFund fails with the following error:

```
java.lang.AssertionError: Unexpected exception thrown: SECJ0053E:
Authorization failed for ??? while invoking
(Beans)DonateEAR#DonateEJB.jar#DonateBean
donateToFund:vacation.entities.Employee,donate.entities.Fund,int:1 null
```

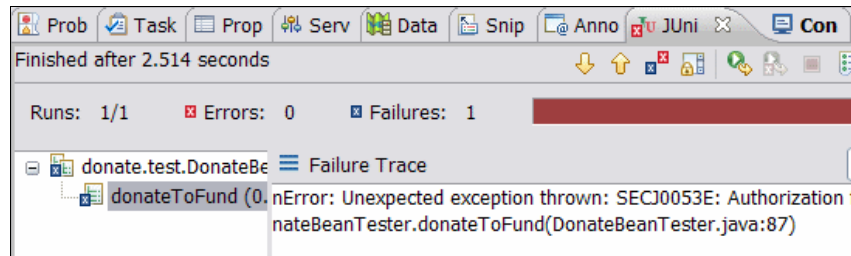




Figure 10-13 donateToFund errors

### Off course?

If the access succeeds, then the method is not protected by a `@RolesAllowed` annotation. Verify the changes made in 10.5.1, “Restrict access to the Donate session EJB (declarative)” on page 373.

3. Change `DonateBeanTester` to access the `DonateBean` as `DNARMSTRONG`.
  - a. In the Enterprise Explorer, open **DonateBeanTester.java** (in `DonateUnitTester/src/donate.test`).
  - b. In the `init` method, insert blank lines after the initial context line.
 

```
ctx = new InitialContext(props);
```
  - c. Insert the **F10.2 DonateBeanTester Security** snippet (Example 10-2).
  - d. Organize imports **Organize imports (Ctrl+Shift+O)** to resolve:
 

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import com.ibm.websphere.security.auth.WSSubject;
import com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl;
```
  - e. Change the **propDir** variable to match the profile directory used on your system.
    -  `C:\IBM\SDP\runtimes\base_v7\profiles\ExperienceJEE`
    -  `/opt/IBM/SDP/runtimes/base_v7/profiles/ExperienceJEE`
  - f. Change the "xxxxxxx" password value to match the password that you used for `DNARMSTRONG` in 10.3, “Preliminary security setup” on page 365.
 

```
LoginContext lc = new LoginContext("WSLogin",
    new WSCallbackHandlerImpl("DNARMSTRONG", "xxxxxxx"));
```
  - g. Save and close `DonateBeanTester.java`.

```
...
ctx = new InitialContext(props);

/** Thin client security configuration
String propDir =
"file:C:/IBM/SDP/runtimes/base_v7/profiles/ExperienceJEE/properties/";
System.setProperty("com.ibm.SSL.ConfigURL", propDir +
    "ssl.client.props");
System.setProperty("java.security.auth.login.config", propDir +
    "wsjaas_client.conf");
System.setProperty("com.ibm.CORBA.ConfigURL", propDir +
    "sas.client.props");
ctx.lookup("");

LoginContext lc = new LoginContext("WSLogin",
    new WSCallbackHandlerImpl("DNARMSTRONG", "xxxxxxx"));
lc.login();

System.out.println("subject=" + lc.getSubject().toString());

WSSubject.setRunAsSubject(lc.getSubject());

donateBean = (DonateBeanRemote)
    ctx.lookup("donate.ejb.interfaces.DonateBeanRemote");
...
...
```

---

### Behind the scenes:

This snippet performs the following key activities:

- Defines the SSL, JAAS and SAS (secure authentication services) properties needed to access the server (using the default files generated in the profile directory):

```
System.setProperty("com.ibm.SSL.ConfigURL", propDir +  
    "ssl.client.props");  
System.setProperty("java.security.auth.login.config", propDir +  
    "wsjaas_client.conf");  
System.setProperty("com.ibm.CORBA.ConfigURL", propDir +  
    "sas.client.props");
```

- Connects to the InitialContext to load the default realm and bootstrap host/port information that is needed by the SecurityServer.

```
ctx.lookup("");
```

- Create the login context, login, and set the resulting authenticated subject as the default subject for this execution thread:

```
LoginContext lc = new LoginContext("WSLogin",  
    new WSCallbackHandlerImpl("DNARMSTRONG", "xxxxxxx"));  
lc.login();  
  
final Subject subject = lc.getSubject();  
System.out.println("subject=" + subject.toString());  
  
WSSubject.setRunAsSubject(subject);
```

4. Verify that the access succeeds with the updated JUnit test.
  - a. In the Project Explorer, select **DonateBeanTester** (in `DonateUnitTester/src/donate.test`), right-click and select **Run As → JUnit Test**.
  - b. In the JUnit view, verify that the call to `donateToFund` succeeded.

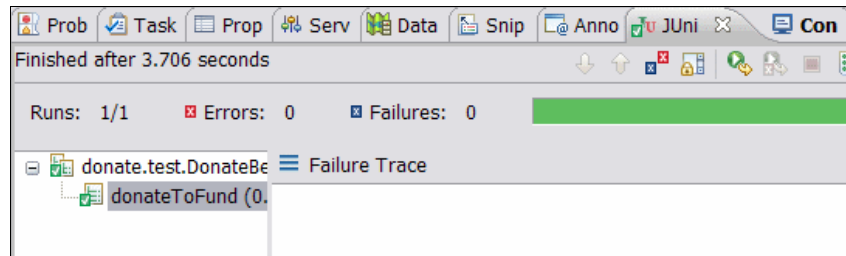


Figure 10-14 *donateToFund success*

### Off course?

- If the access fails with the following error, the most likely cause is that the project does not have the required `com.ibm.ws.ejb.thinclient_7.0.0.jar` file needed to enable the client to interact with the server JAAS support. Refer to 5.8.1, “Create the JUnit project” on page 180 for details.

```
java.lang.AssertionError: Unexpected exception thrown: SECJ0053E:
Authorization failed for ??? while invoking
(Beans)DonateEAR#DonateEJB.jar#DonateBean ...
```

- If the access fails with the following error, then the user mapping is not configured properly in the `DonateEAR` binding configurations. Refer to 10.5.3, “Configure `DonateEAR` roles and role mapping” on page 380.

```
java.lang.AssertionError: Unexpected exception thrown: SECJ0053E:
Authorization failed for defaultWIMFileBasedRealm/DNARMSTRONG
while invoking (Beans)DonateEAR#DonateEJB.jar#DonateBean
donateToFund:vacation.entities.Employee,donate.entities.Fund,int:1
Subject:
Principal: defaultWIMFileBasedRealm/DNARMSTRONG
...
is not granted any of the required roles: DMANAGERS_ROLE
DUSERS_ROLE
```

## 10.5.5 Test the EJB declarative security using the UTC

In this section you utilize the Universal Test Client to verify that the `Donate` session EJB cannot be accessed by an unauthenticated user. You will do this using the Universal Test client.

The UTC invokes EJBs without a user context, meaning that access to the secured `DonateBean` EJB will fail because the active user context

(UNAUTHENTICATED) does not belong to the DUSERS\_ROLE or DMANAGERS role.

**Behind the scenes:** The Universal Test Client does require the user to log on to the system, but that is only used to access the WebSphere Application Server JNDI directory and classpath.

However, to invoke an EJB via the UTC, we must first log on with a userid that has the required permissions to load the classes used by the EJBs. This user logon is used strictly for that purpose, and is not used as the user principal for the EJB, which instead runs as UNAUTHENTICATED.

he javaeeadmin has these required permissions to access the classes because it is the default administrator,

However DNARSMTRONG, DCBROWN, and DGADAMS do not, If you attempted to use the UTC using any of these users you will not be able to load any of the EJBs and will observe the following error in the Console view:

```
[7/15/09 11:33:03:437 EDT] 00000419 RoleBasedAuth A SECJ0305I: The
role-based authorization check failed for admin-authz operation
AppManagement:listApplications:java.util.Hashtable:java.lang.String. The
user DNARMSTRONG (unique ID:
user:defaulttwimfilebasedrealm/uid=dnarmstrong,o=defaulttwimfilebasedrealm)
was not granted any of the following required roles: operator, deployer,
configurator, monitor, administrator, adminsecuritymanager, auditor.
```

You can resolve this by adding these users to any of these groups via the Admin Console (under Users and Groups → Administrative users). This will allow these users to successfully load the DonateBean EJB. However, the actual invocation will still fail because the UTC will still invoke the EJB as UNAUTHENTICATED.

1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Restart the **UTC** by selecting the server, right-clicking and selecting **Universal test client** → **restart**.
  - a. Log in using the **javaeeadmin** User ID and password that you specified in section 3.4, “Create the ExperienceJEE server profile” on page 62.
3. In the resulting **Test Client**, in the upper left select **JNDI Explorer**.
4. In the **JNDI Explorer** pane (in the right) open the reference to the entity EJB remote home interface by selecting **ejb** → **DonateEAR** → **DonateEJB.jar** → **DonateBean#donate.ejb.interfaces.DonateBeanRemote** (**donate.ejb.interfaces.DonateBeanRemote**).



5. In the **EJB Beans** section (on the left), select **EJB Beans** → **DonateBeanRemote** → **DonateBeanRemote** → **String donateToFund**
6. In the resulting right panel:
  - a. Under the **Value** column in the first row (for the employee ID) enter **1**.
  - b. Under the **Value** column in the second row (for the fund name) enter **DonationFund**.
  - c. Under the **Value** column in the third row (for the transfer amount) enter **1**.
  - d. Click **Invoke**.
7. Observe the failure message indicating that UNAUTHENTICATED does not have any of the required roles.

Parameters

java.lang.String donateToFund(int, String, int)

Parameter	Value
int:	1
String:	DonationFund <span>Objects</span>
int:	1

Invoke

Results from donate.ejb.interfaces.\_DonateBeanRemote\_Stub.donateToFund()  
 SECJ0053E: Authorization failed for /UNAUTHENTICATED while invoking (Bean) DonateEAR#DonateEJB.jar#DonateBean donateToFund:int,java.lang.String,int:1 Subje  
 Principal: /UNAUTHENTICATED Public Credential:  
 com.ibm.ws.security.auth.WSCredentialImpl@12a012a0 is not granted any of the  
 required roles: DMANAGERS\_ROLE DUSERS\_ROLE

Figure 10-15 Error message indicating the unauthenticated user was denied access

### Off course?

If the access succeeds, then the method is not protected by a `@RolesAllowed` annotation. Verify the changes made in 10.5.1, “Restrict access to the Donate session EJB (declarative)” on page 373.

## 10.5.6 Test the Web declarative security

In this section you verify that the web security mechanisms operate correctly. Table 10-2 summarizes the assignments that you made in the previous sections and links the Java EE roles to the file registry groups and the file registry users.

Table 10-2 Summary of previous assignments

Java EE role	Group	User members
DUSERS_ROLE	DUSERS	DCBROWN, DNARMSTRONG
DMANAGERS_ROLE	DMANAGERS	DNARMSTRONG
ALLAUTHENTICATED_ROLE	ALLUSERS	DCBROWN, DNARMSTRONG, DGADAMS

Testing occurs through an external browser because the workbench browser keeps credentials once established, making it difficult to validate security:

1. Test the access using the DGADAMS user ID:
  - a. Open a browser and type the following Web address:  
<http://localhost:9080/DonateJSFWeb/faces/FindEmployee.jsp>
  - b. At the Connect to localhost pop-up, login as DGADAMS using the password defined in 10.3, "Preliminary security setup" on page 365.

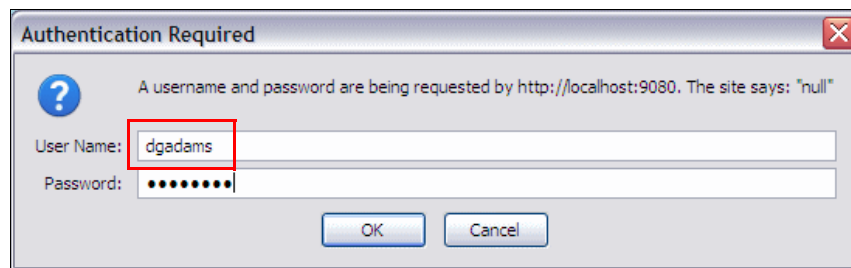


Figure 10-16 Login with a new ID

- c. The initial access to FindEmployee.jsp succeeds because the declarative security for this page allows access by anyone belonging to the ALLAUTHENTICATED\_ROLE, and the group ALLUSERS is assigned to that role (and all users are assigned to the ALLUSERS group).

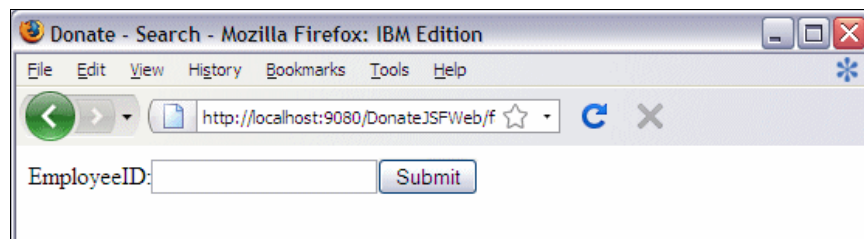


Figure 10-17 The log in succeeds

### Off course?

If you do not see the login pop-up, ensure that you correctly created the Web declarative security in 10.5.2, “Restrict access to Web pages via declarative security” on page 375 for the base pages.

2. Enter a valid employee number (such as 1) in the Employee ID field, and retrieve an employee record (using DGADAMS) by clicking **Submit**. The following failure message appears in the browser.

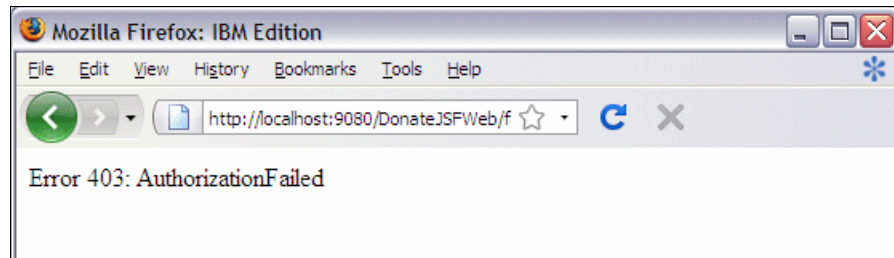


Figure 10-18 Authorization fails

The access to the secondary elements, in this case the `EmployeeDetails.jsp` page, fails because that resource allows access by anyone belonging to the `DUSERS_ROLE` and `DMANAGERS_ROLE` roles, and `DGADAMS` is assigned to neither of these roles.

### Off course?

If this access works, and you in fact see the `EmployeeDetails.jsp` page, ensure that you correctly created the Web declarative security in 10.5.2, “Restrict access to Web pages via declarative security” on page 375 for the Result pages.

3. Test the access using the `DNARMSTRONG` user ID:
  - a. Close the browser.
  - b. Open a new browser and use the same Web address:  
<http://localhost:9080/DonateJSFWeb/faces/FindEmployee.jsp>
  - c. Login as `DNARMSTRONG`.

As previously, the initial access to `FindEmployee.jsp` succeeds because the declarative security for this page allows access by anyone belonging to the `ALLAUTHENTICATED_ROLE` role.

4. Test access to the DonationConfirmation page.
  - a. Enter an employee number of **2**, and click **Submit**. The access to the secondary element (EmployeeDetails.jsp) succeeds because this resource allows access by anyone belonging to the DUSERS\_ROLE and DMANAGERS\_ROLE roles, and DNARMSTRONG is assigned to the DMANAGERS\_ROLE group that is assigned to the DMANAGERS\_ROLE role.

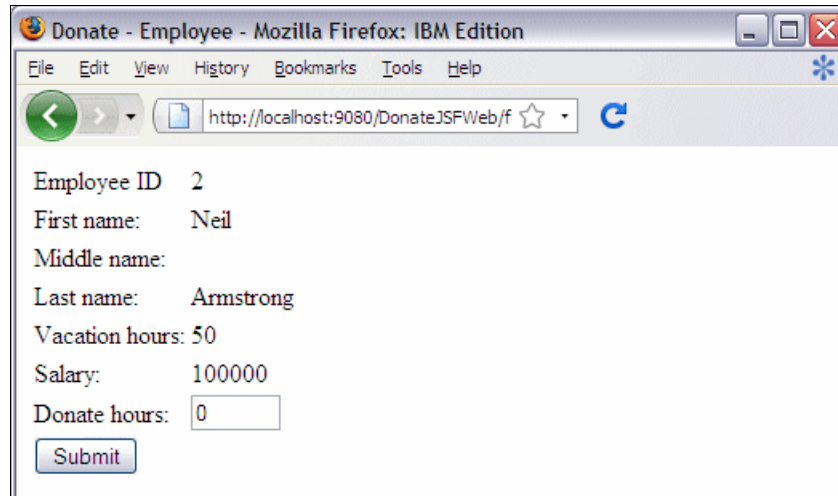


Figure 10-19 Authorization succeeds

- b. Type a Donation hours amount (for example, **2**), and click **Submit**. The access to the secondary elements (the DonateBean session EJB and DonationConfirmation.jsp) succeeds because they both allow access by anyone belonging to the DUSERS\_ROLE and DMANAGERS\_ROLE role, and DNARMSTRONG is assigned to both of these roles.

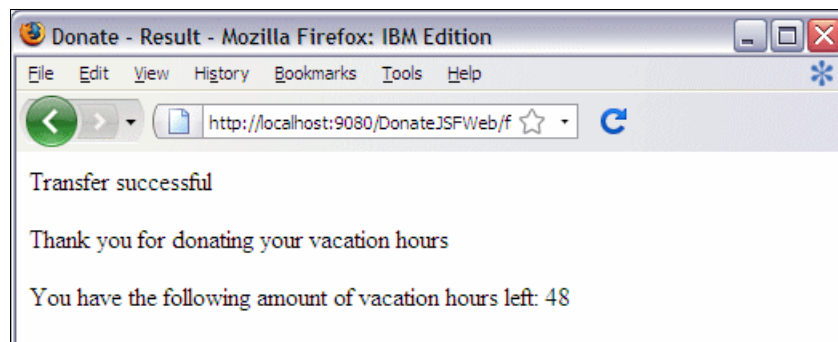


Figure 10-20 Authorization succeeds

### Off course?

If you receive an error on the donation action, ensure that you correctly created the EJB declarative security in 10.5.1, “Restrict access to the Donate session EJB (declarative)” on page 373.

### Extra credit:

Optionally try the foregoing steps using the DCBROWN user ID. The access should succeed because DCBROWN belongs to the DUSERS\_ROLE role.

## 10.6 Implement and test programmatic security

Programmatic security is implemented in user code. It references the role definitions in the deployment descriptors:

- ▶ To illustrate programmatic security in a session EJB, we hide the employee salary for non-managers in the `EmployeeFacade` session EJB.
- ▶ To illustrate programmatic security in a Web page, we hide the display of the salary for non-managers in the `EmployeeDetails.jsp`.

### 10.6.1 Add clone method to the Employee entity

We want to mask (hide) the salary of an employee to non-managers. When we retrieve an employee object in the `EmployeeFacade`, we have to ensure that masking the salary is not propagated to the back-end database.

For example, if we change this field directly in the employee object in the `findEmployeeById` method, the change would be automatically committed to the database.

We have several options:

- ▶ Create a new object, for example, `EmployeePrime`, and copy all the fields.
- ▶ In the `EmployeeFacade`, instantiate a new `Employee` object and copy all the fields.
- ▶ In the `Employee` class, add a `clone` method (to create a copy of the object). Then, in the `EmployeeFacade`, call this method to create a copy and mask/null the one field (salary). This involves changes in two methods, but the least amount of code.

Here, we use the `clone` method:

1. In the Enterprise Explorer, open **Employee.java** (in `DonateJPAEmployee/src/vacation.entities`).
2. In the Employee editor:

- a. Add the **Cloneable** interface to the class definition:

```
public class Employee implements Serializable, Cloneable {
```

- b. Implement the `clone` method (required by the interface) by inserting the following method after the class definition.

```
public class Employee implements Serializable, Cloneable {  
  
    public Employee clone() throws CloneNotSupportedException {  
        return (Employee) super.clone();  
    }  
    ...  
}
```

- c. Save and close `Employee.java`.

#### Behind the scenes:

The `Cloneable` interface copies the entire content of the original object to the new object. This new object is not linked to the original object, so any changes to this object are not reflected in the back-end database.

## 10.6.2 Implement programmatic security in the `EmployeeFacade`

In this section, we update the `EmployeeFacade` session EJB to restrict access to the salary field to members of the `DMANAGERS_ROLE` role:

1. In the Enterprise Explorer, open **EmployeeFacadeBean.java** (in `DonateEJB/ejbModule/vacation.ejb.impl`).
2. In the `EmployeeFacadeBean` editor:

- a. Add the following line above the class definition to define the `DMANAGERS_ROLE` role within this class:

```
@Stateless  
@Local( { EmployeeFacadeInterface.class })  
@DeclareRoles( { "DMANAGERS_ROLE" })  
public class EmployeeFacade implements EmployeeFacadeInterface, ... {  
    .....  
}
```

- b. Add the following lines below the class definition to define the session context object that will be used to determine if the user is in the `DMANAGERS_ROLE` role:

```

public class EmployeeFacade implements EmployeeFacadeInterface,
   EmployeeFacadeRemote {

    @Resource
    private SessionContext ctx;

```

- c. Locate the `findEmployeeById` method, comment the return statement and insert the **F10.3 EmployeeFacade Salary Hide** snippet from the Snippets view:

```

public Employee findEmployeeById(int id) throws Exception {
    Employee employee = em.find(Employee.class, id);
    if (employee == null)
        throw new Exception("Employee with id " + id + " not found");
    // return employee;
    System.out.println("EmployeeFacade: User id="
        + ctx.getCallerPrincipal().getName());
    System.out.println("EmployeeFacade: isCallerInRole="
        + ctx.isCallerInRole("DMANAGERS_ROLE"));
    Employee clone = employee.clone();
    if (!ctx.isCallerInRole("DMANAGERS_ROLE"))
        clone.setSalary(null);
    return clone;
}

```

- d. Organize imports (**Ctrl+Shift+O**) to resolve:

```

import javax.annotation.Resource;
import javax.annotation.security.DeclareRoles;
import javax.ejb.SessionContext

```

- e. Save and close `EmployeeFacade.java`.

**Behind the scenes:** The salary field is cleared from the employee object if the user does not belong to the `DMANAGERS_ROLE`.

Note that this has implications in how the returned employee object is used. Because the salary field has been cleared (null), any future calls that use this object would store the null value in the database.

For example, the current `addHours` method in the `EmployeeFacade` is coded correctly to avoid propagating a null salary field to the database:

```

public Employee addHours(Employee employee, int hours) {
    employee = em.find(Employee.class, employee.getEmployeeId());
    employee.setVacationhours(employee.getVacationhours() + hours);
    em.merge(employee);
    return employee;
}


```

### 10.6.3 Test programmatic security in a session EJB

We use the JSF Web application to test programmatic security in the session bean:

1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Test the access using the DNARMSTRONG user ID:
  - a. Open a new browser, and use the following URL:  
<http://localhost:9080/DonateJSFWeb/faces/FindEmployee.jsp>
  - b. Log in as DNARMSTRONG.
  - c. In the resulting page, retrieve record 2.

Note that salary information is still visible because DNARMSTRONG belongs to the DMANAGERS group, which is assigned to the DMANAGERS\_ROLE role that has visibility into this information.



Employee ID 2  
First name: Neil  
Middle name:  
Last name: Armstrong  
Vacation hours: 48  
Salary: 100000  
Donate hours: 0  
Submit

Figure 10-21 Authorization succeeds

3. Test the access using the DCBROWN user ID:
  - a. Close the browser, and open a new browser with the same URL:
  - b. Log in as DCBROWN.
  - c. In the resulting page, retrieve record 2.

Note that salary information is masked because DCBROWN belongs to the DUSERS group, which is assigned to the DUSERS\_ROLE role, and this role does not have visibility to this information.



Employee ID	2
First name:	Neil
Middle name:	
Last name:	Armstrong
Vacation hours:	48
Salary:	
Donate hours:	0
<input type="button" value="Submit"/>	

Figure 10-22 Authorization fails for the salary information is masked

4. Close the browser.
5. In the workbench, switch to the Console view and notice the output messages in the ExperienceJEE Test Server log:

```
EmployeeFacade: User id=DNARMSTRONG
EmployeeFacade: isCallerInRole=true
EmployeeFacade: User id=DCBROWN
EmployeeFacade: isCallerInRole=false
```

## 10.6.4 Implement programmatic security in a Web application

The EJB programmatic security eliminates the value of the salary field for non-authorized users, but the front-end JSPs still contain the name of the field.

We can use programmatic security in the `EmployeeDetails.jsp` to eliminate the display of the salary field for non-authorized users. This helps address the security concern where knowledge of the field is considered to be a security concern, in addition to the actual values:

1. In the Enterprise Explorer, open **DonateHoursManagedBean.java** (in `DonateJSFWeb /Java Resources: src/donate.jsf.bean`).
2. In the `DonateHoursManagedBean` editor:

- a. Add an `isManager` method before the closing brace, using the **F10.4 Donate Managed Bean isManager** snippet from the Snippets view:

```
public boolean isManager() {
    FacesContext context = FacesContext.getCurrentInstance();
    return context.getExternalContext().isUserInRole("DMANAGERS_ROLE");
}
```

- b. Save and close `DonateHoursManagedBean.java`.

### Behind the scenes:


Similar to the `isCallerInRole` method used with EJBs, the `isUserInRole` checks to see if the user is in the specified role and returns the appropriate true or false value.

This method is put into a Faces managed bean because the page does not have direct access to the `FacesContext` instance that contains this method.

3. In the Enterprise Explorer, select **EmployeeDetails.jsp** (in `DonateJSFWeb/WebContent`), right-click and select **Open With** → **Page Designer**.
4. In the `EmployeeDetails.jsp` editor select the **Split** tab and:
  - a. In the upper split window (the design view) click on **Salary**. Note that the source is automatically located in the lower split window (the source view).



Figure 10-23 `EmployeeDetails.jsp`

- b. Go to the Properties view, and select the Attributes icon [  ] in the upper right of the view action bar.

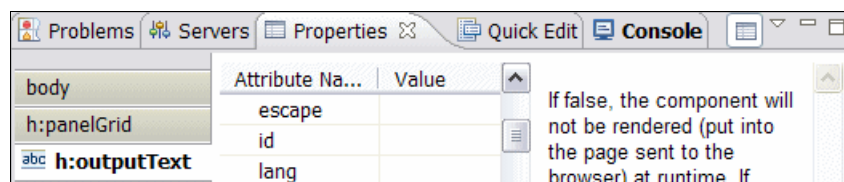


Figure 10-24 Select the attributes

- c. Select the **rendered** property, and for the value select the pull-down on the right and select **Compute**.

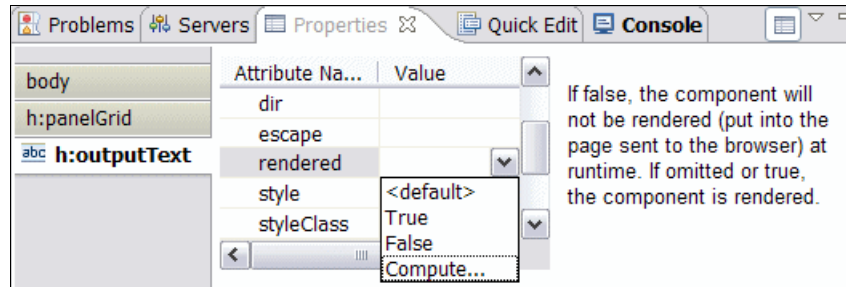


Figure 10-25 Select the rendered property

- d. At the Select Page Data Object pop-up, ensure that you are on the DataObject tab, select **donateHoursManagedBean ()** → **manager (boolean)** and click **OK**.

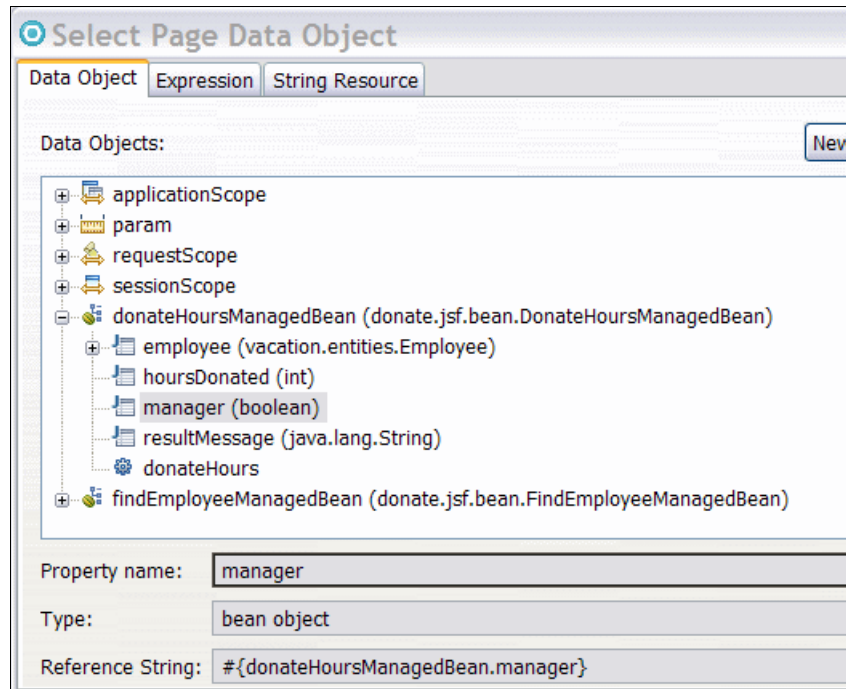


Figure 10-26 Set the rendered property

- e. Back in the Properties view, note that the rendered value is set to the reference string selected above (`{donateHoursManagedBean.manager}`)

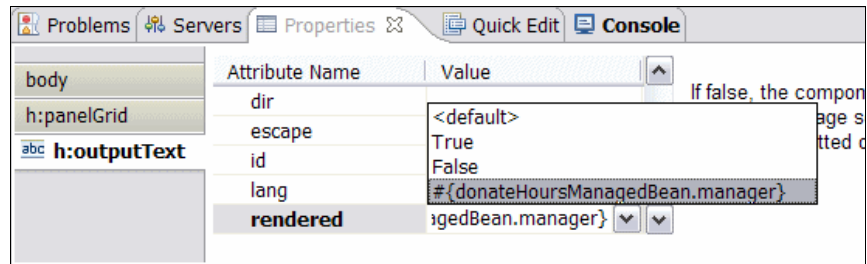


Figure 10-27 New setting for the rendered value

- f. Repeat this change for the output field of the salary (`{salary}`).
- g. Back in the `EmployeeDetails.jsp` editor, note that the rendered field is set for both the Salary label and value output fields

```
<h:outputText styleClass="outputText" id="text4" value="Salary: "
    rendered="{donateHoursManagedBean.manager}">
</h:outputText>
<h:outputText styleClass="outputText" id="text3"
    value="{findEmployeeManagedBean.employee.salary}"
    rendered="{donateHoursManagedBean.manager}">
</h:outputText>
```

- h. Save and close `EmployeeDetails.jsp`.

**Behind the scenes:** If the caller belongs to the `DMANAGERS_ROLE` role, then the salary field is displayed as before. If the user does not belong to the `DMANAGERS_ROLE` role, then the salary field is skipped.

The Faces rendered property binding of `{managedbean.<someproperty>}` indicates that a method named `is<SomeProperty>` is called to determine if the field is displayed. In our case, `isManager` in `DonateHoursManagedBean` is invoked.

## 10.6.5 Test programmatic security in a Web application

In this section you test the combined Web and EJB programmatic security.

1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Run the Web application with user ID DNARMSTRONG (always close the browser and open a new browser) and lookup employee 2. The salary label and field are displayed.
3. Run the Web application with user ID DCBROWN and lookup employee 2. The salary label and field is not displayed.

DNARMSTRONG	DCBROWN
Employee ID 2	Employee ID 2
First name: Neil	First name: Neil
Middle name:	Middle name:
Last name: Armstrong	Last name: Armstrong
Vacation hours: 48	<u>Vacation hours: 48</u>
Salary: 100000	Donate hours: 0
Donate hours: 0	Submit
Submit	

Figure 10-28 Salary information display results

**Extra credit:** Rerun the test for DNARMSTRONG and lookup employee 1. Note that the Salary label is still displayed even though the salary value is blank.

## 10.7 Update the Java EE application client for security

The Java EE application client is subject to the same authentication and authorization requirements as the Web front end. There are three ways to supply the authentication information:

- ▶ Through a pop-up prompt, as you already experienced in section 9.6, “Test the application client” on page 350.
- ▶ Through a properties file, such as `sas.client.props`.
- ▶ Through programmatic login.

This example utilizes the properties file methods.

## 10.7.1 Demonstrate the existing behavior

In Chapter 9, “Create the application client” on page 345 you dismissed an authentication prompt when invoking the DonateClient application client because the user information was not needed. In this section, you will verify that the access will fail when you attempt to invoke the client without credentials, and you will verify that access will succeed when you provide the requested credentials.

This section uses the DonateClient launch configuration created in 9.6, “Test the application client” on page 350.

1. Demonstrate that DonateClient will fail to access the Donate EJB session bean when invoked without credentials:
  - a. From the workbench action bar select **Run** → **Run Configurations**,
  - b. At the Run Configurations pop-up, select **WebSphere Application Server v7.0 Application Client** → **DonateClient** and click **Run**.
  - c. At the Login at the Target Server pop-up, click **Cancel**.
  - d. Switch to the DonateClient Console view output and verify that the application client failed. The failure stack trace will exceed 50 lines, and you are looking for these specific messages:

```
...
Caused by: javax.ejb.EJBAccessException:
>> SERVER (id=4773e3aa, host=testwin.demo.com) TRACE START:
>> javax.ejb.EJBAccessException: SECJ0053E: Authorization failed
for ??? while invoking (Bean)DonateEAR#DonateEJB.jar#DonateBean
donateToFund:int,java.lang.String,int:1 null
>> at
com.ibm.ws.security.core.SecurityCollaborator.performAuthorization(Secur
ityCollaborator.java:672)
```

2. Demonstrate that DonateClient will successfully access the Donate EJB session bean when invoked with credentials:
  - a. From the workbench action bar select **Run** → **Run Configurations**,
  - b. At the Run Configurations pop-up, select **WebSphere Application Server v7.0 Application Client** → **DonateClient** and click **Run**.
  - c. At the At the Login at the Target Server pop-up enter the DNARMSTRONG userid and password that you configured in 10.3, “Preliminary security setup” on page 365, and click **OK**.
  - d. Switch to the DonateClient Console view output and verify that the application client succeeded:

```
...
CWSCL0014I: Invoking the Application Client class Main
Employee Number = 1
```

```
Fund                = DonationFund
Donation Amount = 1
Donation result = Transfer successfu
```

## 10.7.2 Create a customized properties file

1. : From a command prompt, copy:  
`C:\IBM\SDP\runtimes\base_v7\profiles\ExperienceJEE\properties\sas.client.props`  
to:  
`C:\7827code\ejbclient.props`
2. Edit **ejbclient.props** and set the following keywords:
  - `com.ibm.CORBA.loginSource=properties`
  - ...
  - `com.ibm.CORBA.loginUserid=DNARMSTRONG`
  - `com.ibm.CORBA.loginPassword=set to the proper password`
3. Save and close `ejbclient.props`.

**Extra credit:** The WebSphere **PropFilePasswordEncoder** utility encrypts the passwords in property files.

For example, if you ran the utility under windows:

```
C:
CD \7827code
C:\IBM\SDP\runtimes\base_v7\profiles\ExperienceJEE\bin\PropFile
PasswordEncoder.bat ejbclient.props -SAS
```

The `com.ibm.CORBA.loginPassword` keyword value is encrypted:

```
com.ibm.CORBA.loginPassword={xor}JycnJycnJyc\=
```

## 10.7.3 Update and test the Java EE application client test configuration

In this section you update the `DonateClient` run configuration to reference the updated property file.

1. From the workbench action bar, select **Run** → **Run Configurations**
2. At the Run Configuration pop-up:
  - a. Under **Configurations**, select the **WebSphere v7.0 Application Server v7.0 Application Client** → **DonateClient** launch configuration.

- b. Switch to the Arguments tab, and change this value under **VM arguments** from:

```
-Dcom.ibm.CORBA.ConfigURL=file:C:/IBM/SDP/runtimes/base_v7/profiles/ExperienceJEE/properties/sas.client.props
```

to

```
-Dcom.ibm.CORBA.ConfigURL=file:C:/7827Code/ejbclient.props
```

- c. Click **Apply**, and then click **Run**.
- d. View the DonateClient output from the **Console** view. Note that the client completes the call to the Donate EJB and that no user prompt is presented.

```
Employee Number = 1
Fund             = DonationFund
Donation Amount = 1
Donation result = Transfer successful
```

## 10.8 Explore!

In this section we describe a few Explore activities.

### 10.8.1 Java EE Security

Java EE security contains many aspects, and addressing all of them would require a separate book (and in fact that was done; see the reference list in the Learn section). Some of the other security aspects include:

- **Java 2 Security:** This is a security mechanism provided by the core J2SE environment to manage an application's access to operating system resources, such as files and network resources. The policies are defined in a text policy file, based on the calling code (not users), and are enforced by the runtime.

Note that Java 2 Security grants access. After Java 2 Security is enabled, by default all access is denied unless explicitly granted by a policy file. This is the opposite of Java EE application role based security, where by default access is allowed unless explicitly denied.

- **Java Authorization Contract for Containers (JACC):** JACC provides a common interface for connecting Java EE application servers with an external authorization provider (such as IBM Tivoli® Access Manager). This allows the Java EE application server authorization definitions to be managed by a centralized authorization system instead of by an internal Java EE unique mechanism.



JACC requires that Java EE containers be configurable to point to an external authorization provider, both for propagating policy information when installing and removing a Java EE application and when making runtime authorization decisions.

Additional Learn resources are available at the following Web addresses:

- ▶ WebSphere InfoCenter: Implementing single sign-on to minimize Web user authentications:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/tsec\\_msso.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/tsec_msso.html)
- ▶ WebSphere InfoCenter: Java 2 Security:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/csec\\_rsecmgr2.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/csec_rsecmgr2.html)
- ▶ WebSphere InfoCenter: JACC support in WebSphere Application Server:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/csec\\_jaccsupport.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/csec_jaccsupport.html)





## Part 3

# Web services

In Part 3, we extend the core Java EE application to support Web services:

- ▶ Chapter 11, “Create the Web service” on page 409: Use the tooling wizards to expose the DonateBean session EJB as a Web service, and create a Web service client to access this Web service.
- ▶ Chapter 12, “Implement security for the Web service” on page 447: Update the Web service and the Web service client to support Web services security using user ID and password for authentication.





## Create the Web service

In this chapter, we discuss Web services and the support provided by Java EE.

We develop a bottom-up Web service from the DonateBean session bean, and create a Web service client for testing.

We also provide examples of a Web services with complex data types, fault handling, and develop a top-down Web service.

## 11.1 Learn!

Figure 11-1 shows Web services and the support provided by Java EE.

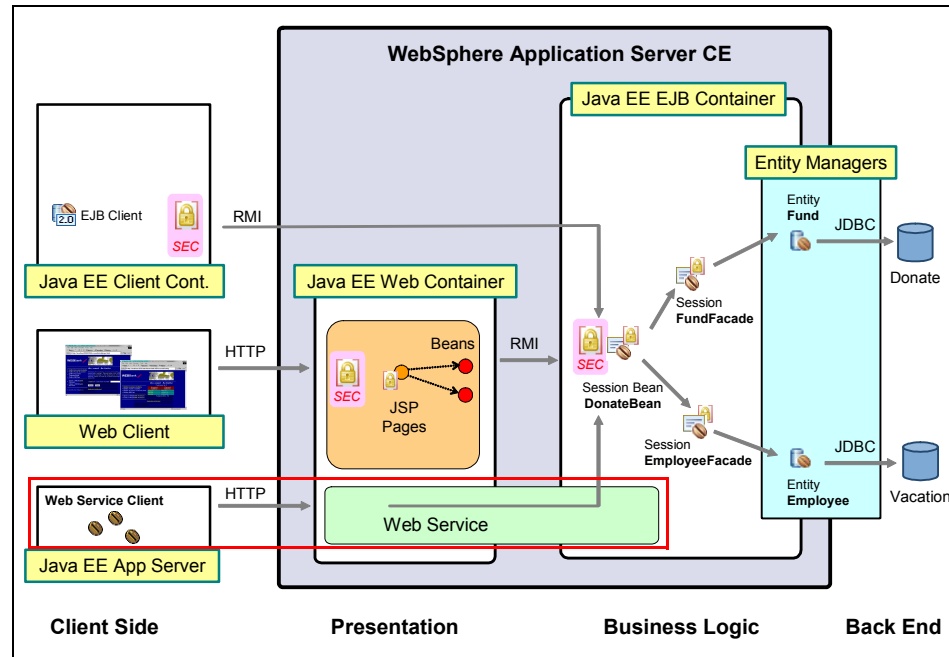


Figure 11-1 Donate application: Web service

Web services provide a standards-based implementation for process-to-process communication between two presumably disparate systems: one providing access to a function through a service, and one consuming this function through a client. Key standards include these:

- ▶ Hypertext Transfer Protocol (**HTTP**) is the most common transport protocol to connect the service and the client.

However, current specifications allow providing services over alternate protocols (other than HTTP) such as messaging, which effectively extends Web services to provide “asynchronous” operation.

- ▶ eXtensible Markup Language (**XML**) describes the overall content (being delivered over HTTP) in a structured format that is recognized by both the service and the client.

- ▶ **SOAP**<sup>1</sup> messages describe the remote operation and the associated request or response message data in an XML based format. Runtime environments such as Java EE and .NET are the primary producers and consumers of SOAP messages, which are transferred over HTTP.

Thus, a SOAP message is the runtime instance of a Web service request or response.

- ▶ Web Services Description Language (**WSDL**) files describe the remote operation and the associated request or response messages in an XML based format. Development tools are the primary producers and consumers of WSDL documents, which are stored in files.

Thus, a WSDL file is the development time representation of a Web service.

- ▶ Universal Description, Discovery, and Integration (**UDDI**) is an optional component that stores and retrieves WSDL documents. This allows the developers of Web services to publish WSDL documents to a directory, and the developers of Web services clients to search this directory for the desired service.

UDDI can also be used at runtime to dynamically discover part or all of the information about a Web service:

- *Partial dynamic* operation is querying the UDDI registry for a list of endpoints that implement the Web service, and then selecting one of these. The endpoint is effectively the Web address for the Web service, updated for the hostname on which the Web service is deployed.
- *Full dynamic* operation is querying the UDDI registry for a complete WSDL that represents a desired type of service, and then dynamically creating and executing a Web service client, potentially selecting different messages, operations, or transports (HTTP or messaging).

Practically, most implementations use *partial dynamic* operation. They do so because although *full dynamic* operation is technically feasible, it provides very poor performance (because you generate, compile, and execute a new Web service client each time), it is complex to code to, and it can potentially generate a large number of runtime errors.

The UDDI specifications supported a publicly accessible Universal Business Registry (UBR) in which a naming system was built around the UDDI-driven service broker. IBM, Microsoft, and SAP announced they were closing their public UDDI nodes in January 2006.

---

<sup>1</sup> Note that SOAP once stood for *Simple Object Access Protocol*, but this acronym was dropped with Version 1.2 of the standard, as it was considered to be misleading. Version 1.2 became a W3C Recommendation on June 24, 2003. The acronym is sometimes confused with SOA (service-oriented architecture), however SOAP is quite different from SOA.

In many companies, UDDI has been replaced by private registries, such as the IBM WebSphere Service Registry and Repository (WSRR), which helps maximize the business value of your service oriented architecture (SOA). WSRR is an industry leading solution that enables you to easily and quickly publish, find, enrich, manage, and govern services and policies in your SOA

An alternative to UDDI is the Web Services Inspection Language (WSIL), which defines WSDL file locations through XML documents. However, this capability is rarely used because for low-end registry support, most users find it simpler and almost as effective to post their WSDL files on a managed HTTP server.

The technical implementation and capabilities inherent in Web services is not unique, and it has been possible to implement these basic capabilities using off the shelf technologies for years.

However, the value and appeal of Web services in today's environment is the almost universal implementation that enables application developers to implement this capability with minimal coding, supporting access from a wide variety of clients:

- ▶ Almost every major application development vendor has tooling to generate Web services and Web services clients.
- ▶ Almost every major infrastructure vendor, such as those for application servers and databases, has runtimes that support these Web services and Web service clients.

### 11.1.1 Web services and Java EE 5

Java EE 5 provides new features for the development of Web services. It allows to use annotations that can set up the Web service or its client with no need of a deployment descriptor. By simply adding the **@WebService** annotation to a session bean or a POJO, it is enough to expose that as a Web service. This allows the developers to easily define and deploy Web services. However, Java EE 5 keeps the advantages of using deployment descriptors, they can be used by a deployer to augment or override the behavior of the annotations.

These new features are ruled by the following specifications:

- JAX-WS 2.0** The Java API for XML-Based Web Services (JAX-WS), JSR 224, is the fundamental technology for developing Web services in Java EE 5 and is the evolution of JAX-RPC 1.1.
- JAXB 2.0** The mapping between Java artifacts and XML documents are defined by the Java Architecture for XML Binding (JAXB), JSR 222. Differently of the JAX-RPC, JAX-WS 2.0 has delegated the XML binding module to JAXB.



**Annotations** The Web Services Metadata for the Java Platform, JSR 181, specifies the annotations used to define a Web services and its artifacts.

**Java API for XML-Based Web Services (JAX-WS 2.0)**

JAX-WS 2.0 (JSR 224) is the core technology for Web services in Java EE 5 specification. The JAX-WS 2.0 programming model simplifies application development by supporting annotations to define Web services applications and clients from simple POJOs, allowing them to be deployed without deployment descriptors.

JAX-WS supports WS-I Basic Profile 1.1, which ensures that Web services developed with the JAX-WS stack can be consumed by any Web service client that adheres to the WS-I Basic Profile standard.

JAX-WS is the evolution of JAX-RPC 1.1 (JSR 101) and Web Services for J2EE (JSR 109). The main difference regarding JAX-RPC is the message-oriented programming model, that replaces the remote procedure call programming model. It allows JAX-WS to be protocol and transport independent, allowing a direct XML/HTTP binding.

Because JAX-RPC is widely used in the industry, JAX-WS maintains backward compatibility with the JAX-RPC 1.1 specifications, with possible exceptions such as the optional SOAP encoding.

**Note:** The main problem Web services had in its beginning was the development of interoperable services. JAX-RPC was a Java community effort to eliminate this problem and provide a well-known application programming interface for Web service implementations on both the client and server side.

It was broadly adopted in the current Java enterprise servers since 2002. Therefore, JAX-WS backward compatibility is very important to allow new services using the large number of services developed with JAX-RPC.

Table 11-1 provides a list of the main technologies supported by JAX-WS 2.0 and its predecessor JAX-RPC 1.1.

Table 11-1 JAX-RPC versus JAX-WS 2.0 technologies

Technology	JAX-RPC 1.1	JAX-WS 2.0
SOAP	1.1	1.2
XML/HTTP	not supported	supported
WSDL	1.0	1.1

Technology	JAX-RPC 1.1	JAX-WS 2.0
WS-I Basic Profile	1.0	1.1
Annotations	not supported	supported
Mapping model	Internal mapping	Delegated to JAXB 2.0

WebSphere Application Server v7.0 supports JAX-WS 2.1. But it still supports JAX-RPC 1.1.

Additional Learn resources are available at the following Web sites:

- ▶ Design and develop JAX-WS 2.0 Web services:  
<http://www.ibm.com/developerworks/edu/ws-dw-ws-jax.html>
- ▶ Web services hints and tips: JAX-RPC versus JAX-WS, Part 1:  
<http://www.ibm.com/developerworks/webservices/library/ws-tip-jaxwsrpc.html>
- ▶ Java Technology and Web services:  
<http://java.sun.com/javaee/technologies/webservices/>
- ▶ JSR 224: Java API for XML-Based Web Services (JAX-WS) 2.0:  
<http://jcp.org/en/jsr/detail?id=224>
- ▶ JSR 222: JavaTM Architecture for XML Binding (JAXB) 2.0:  
<http://jcp.org/en/jsr/detail?id=222>
- ▶ JSR 181: Web Services Metadata for the JavaTM Platform:  
<http://jcp.org/en/jsr/detail?id=181>

## 11.2 Jump start





If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed all the applications of Part 2, “Core Java EE application” on page 121.

## 11.3 Disable Java EE EJB declarative security

This chapter does not contain the application code required to support a secure session EJB. Therefore, before starting this chapter, we have to disable the Java

EE EJB declarative security. We will re-enable EJB security at the start of Chapter 12, “Implement security for the Web service” on page 447.

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to:
    -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`
  - and execute:
    -  `startNetworkServer.bat`
    -  `startNetworkServers.sh`
  - You will restart the **ExperienceJEE Test Server** after making the changes, so it does matter at this point of the server is running or stopped.
2. Disable declarative security in the DonateBean session EJB:
  - a. In the Java EE Enterprise Explorer, open **DonateBean** (in `DonateEJB/ejbModule /donate.ejb.impl`).
  - b. Before the two `donateToFund` methods, comment out the `@RolesAllowed` statement and add a `@PermitAll` annotation.

```
//@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
@PermitAll
public String donateToFund(Employee employee, .....
    .....


//@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
@PermitAll
public String donateToFund(int employeeId, .....)
```
  - c. Save and close `DonateBean.java`.

#### Workaround:

We should not have to add the `@PermitAll` annotation, because removing the `@RolesAllowed` annotation should leave the method unprotected.

However, because there are still partial security definitions in this class, access to the Web service that we create fails with the following error, unless you add the `@PermitAll` annotation:

```
[Axis2WebServiceContainer] Exception occurred while trying to invoke
service method doService()
org.apache.axis2.AxisFault: javax.ejb.EJBAccessException: Unauthorized
Access by Principal Denied: Unauthorized Access by Principal Denied
```

3. Restart the server to ensure that all active instances of the application are running with security disable.
  - If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
  - If the **ExperienceJEE Test Server** is running, select the server in the Servers view, right-click and select **Restart**.
  - If the server instance starts (restarts) successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.
4. We do not test that declarative security is disabled. We could change and run the DonateBeanTester JUnit test to verify the changes:

```
//LoginContext lc = new LoginContext("WSLogin",
//    new WSCallbackHandlerImpl("DNARMSTRONG", "xxxxxxx"));
//lc.login();
//System.out.println("subject=" + lc.getSubject().toString());
//WSSubject.setRunAsSubject(lc.getSubject());
```

However, we will get errors when testing the Web service if the changes were not made properly.

## 11.4 Create the bottom-up EJB Web service

Web services can be created using two methods: top-down development and bottom-up development. Bottom-up Web service development involves creating a Web service from a Java bean or enterprise bean.

When creating a Web service using a bottom-up approach, first you create a Java bean or EJB and then annotate this Java artifact to expose it as a Web service. JAX-WS 2.0 defines that the Java EE 5 implementation must support mapping Java to WSDL 1.1, which means that the WSDL is automatically generated by the Java EE 5 container.

In this section you create a bottom-up EJB-based Web service from the Donate session bean by adding annotations to the bean class. You only expose the method donateFund that uses only primitive types as parameters:

1. In the Enterprise Explorer, open **DonateBean** (in DonateEJB/ejbModule/donate.ejb.impl).
2. In the editor, add the annotations to the DonateBean class (Example 11-1 on page 418).
  - a. Before the class definition, add a **@WebService** annotation.

- i. You can use the content assistant from the workbench by typing `@W` and press content assist (`Ctrl+space`) scroll down to the bottom and select **WebService(Web Service Template) - javax.ws**. Refer to Example 11-1 below for the inserted annotation.

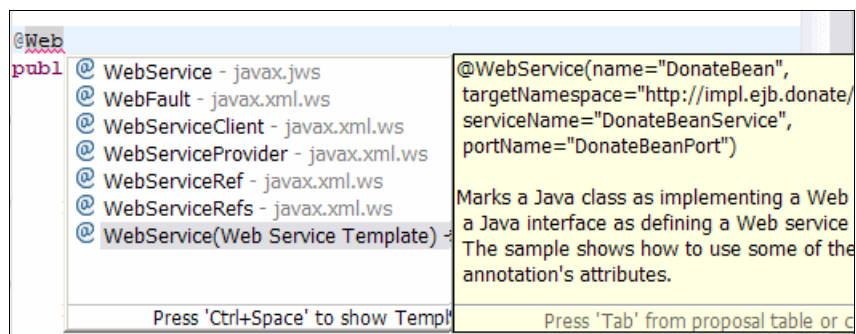


Figure 11-2 Content assist

- b. Before the `donateToFund(int,String,int)` method, add a **@WebMethod** annotation. Be sure to set it at the second `donateToFund` method.
  - i. You can use the content assist using the same steps as above (create a blank line before the method, type `@Webm`, press content assist (`Ctrl+space`), and scroll down and select `WebMethod(Web Service Template) - javax.ws`). Refer to Example 11-1 below for the inserted annotation.
  - ii. Set the empty action value to **urn:donateToFund**.

**Behind the scenes:** ensure that you create a single `@WebMethod` annotation and that this is on the `donateToFund (int,String, int)` method. Otherwise, you will receive the following error when publishing the updated application:

```
Request wrapper bean names must be unique and must not clash with
other generated classes. Class: donate.ejb.impl.DonateBean method
donateToFund(int,java.lang.String,int)()
```

The underlying JAX-WS tooling does not accomodate overloaded methods (methods with the same name) and attempts to generate conflicting classes.

The `WebMethod` annotation operation name field mitigates some aspects of this problem, but does not address the generated names for the wrapper beans and the error still occurs.

- c. Organize the imports (**Ctrl+Shift+O**).
- d. Save and close DonateBean.java.

*Example 11-1 Web service annotations for the session bean*

---

```
@Stateless
@ApplicationException(rollback = true)
@DeclareRoles( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
@WebService(name="DonateBean",
targetNamespace="http://impl.ejb.donate/",
serviceName="DonateBeanService", portName="DonateBeanPort")
public class DonateBean implements DonateBeanInterface, DonateBeanRemote
{
    .....

    //@RolesAllowed( { "DMANAGERS_ROLE","DUSERS_ROLE" })
    @PermitAll
    @WebMethod(operationName="donateToFund", action="urn:donateToFund")
    public String donateToFund(int employeeId, String fundName, int
hours) {
        ...
    }
}
```

---

## Behind the scenes:

The annotations have simplified the complexity to expose session beans or POJO Java classes as Web services. The annotations used to expose the service are:

**@WebService** The `javax.jws.WebService` annotation can be used on a Java interface or a Java class. When used in a Java class, it defines a Web service. When used in a Java interface, it allows the separation of Web service interface and implementation. If no method is explicitly annotated to be exposed as a Web service, by default all the public methods of the annotated class compose the service endpoint interface (SEI).

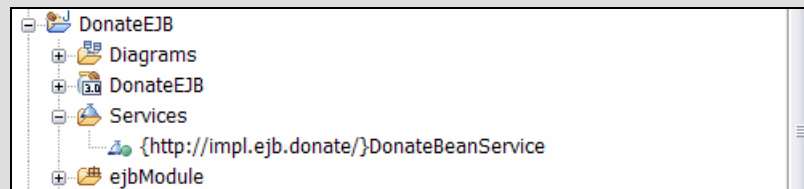
**@WebMethod** The `javax.jws.WebMethod` annotation is used to explicitly define which methods of a class are exposed as Web services.

For example, if the `@WebMethod` annotation was not added to the `DonateBean` session bean, all its public methods would be exposed as services. Because you have explicitly added the annotation to one method only, that method is defined as a service.

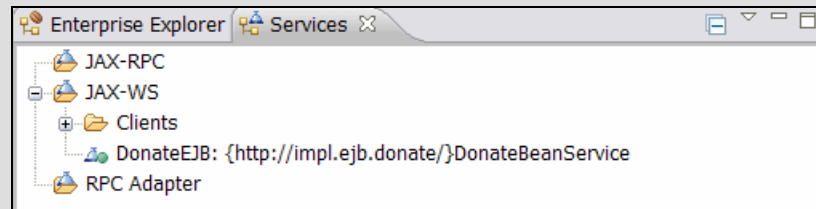
The default arguments available in the template are the following:

<code>operationName</code>	the name of the <code>wsdl:operation</code> matching this method.
<code>action</code>	determines the value of the soap action for this operation.

After annotating the EJB with `WebServices` annotation the project folder `Services` (in the Enterprise Explorer view) displays the service created.



In the Services view there is a list of all services that exist in the workspace grouped by its type. **Deployment descriptors versus annotations:**



### Deployment descriptors versus annotations:

The @WebService annotation is mandatory per JAX-WS 2.0 specifications to expose an EJB session bean as a Web service. Customizations of this annotation, such as Web service endpoint URL, security definitions, the virtual host used by the Web service can also be defined in the OpenEJB deployment descriptor. The deployment descriptor overrides the annotations, allowing the deployer to easily change how the Web service is deployed.

3. Select **DonateBeanService** (in DonateEJB/Services), right-click and select **Create Router Modules (EndpointEnabler)**

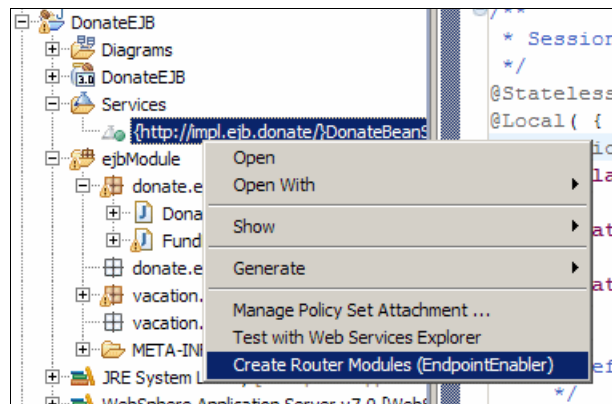


Figure 11-3 Create the router modules



**Behind the scenes:** a router module is needed to allow transport of SOAP messages over HTTP.

The Java EE specification does not define how to expose the HTTP interface for a EJB based Web Service. In WebSphere Application Server a HTTP Router project is required to allow HTTP access to an EJB that is exposed as a WebService.

A router module provides an endpoint for the Web services in a particular enterprise bean JAR module. It can be created using the workbench. If you have an assembled EAR with an EJB exposed as a Web Service without the router project use the `endptEnabler` command to enable the EAR file for serving the EJB thru HTTP.

More information about the `endptEnabler` command can be found in the WebSphere Application Server Information Center:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.iseries.doc/info/seriesnd/ae/twbs\\_endptenablere ndpt.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.iseries.doc/info/seriesnd/ae/twbs_endptenablere ndpt.html)

Other Java EE providers may have different approaches, for example WebSphere Community Edition and Geronimo expose the HTTP interface directly from the EJB module using specific OpenEJB deployment descriptors to define the HTTP mapping.

After you generate the Web service, you can view the router project `web.xml` (in `DonateEJB_HTTPRouter/WebContent/WEB-INF`) for the servlet definitions used to route the requests to the EJB-based Web service:

```
<servlet>
    <servlet-name>donate.ejb.impl.DonateBean</servlet-name>
    <servlet-class>
com.ibm.ws.websvcs.transport.http.WASAxis2Servlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>donate.ejb.impl.DonateBean</servlet-name>
    <url-pattern>/DonateBeanService</url-pattern>
</servlet-mapping>
```

`ibm-webservices-bnd.xml` is created in `DonateEJB/ejbModule/META-INF` to define the binding between the HTTP router module and the EJB project:

```
<com.ibm.etools.webservice.wsbind:WSBinding xmi:version="2.0" ...
xmi:id="WSBinding_1263484961984">
    <routerModules xmi:id="RouterModule_1263484961984"
transport="http" name="DonateEJB_HTTPRouter.war"/>
</com.ibm.etools.webservice.wsbind:WSBinding>
```

4. At the Create Route Project pop-up:
  - a. Enable the **HTTP** binding check box and clear the **JMS** binding check box.
  - b. Set the HTTP Router Project Name to **DonateEJB\_HTTPRouter**
  - c. Click **Finish**.

#### **Behind the scenes:**

The multiprotocol binding support for JAX-WS Web service is a new feature of Application Developer v7.5. The multiprotocol support for JAX-WS is an extension of the JAX-WS programming model, and extends the existing JAX-WS capabilities to support binding types like HTTP (SOAP over HTTP) and JMS (SOAP over JMS).

SOAP over HTTP is the most commonly used Web services binding type. However, if you are looking for a more reliable, scalable, and guaranteed messaging mechanism, consider using SOAP over JMS.

Only SOAP over HTTP is WS-I compliant.

5. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
6. Open the following URL in a browser to view the WSDL file that was generated by the container during the application deployment. Note that the WSDL file will be rendered differently depending on the browser you are using.

[http://localhost:9080/DonateEJB\\_HTTPRouter/DonateBeanService?wsdl](http://localhost:9080/DonateEJB_HTTPRouter/DonateBeanService?wsdl)

**Behind the scenes:** Example 11-2 shows the complete text-view of the WSDL file.

Note that the endpoint address contains the URL for this deployed instance of the Web service. Other instances of the WSDL accessed outside the deployment context often contains the placeholder `REPLACE_WITH_ACTUAL_URL`.

```
...  
...  
    <service name="DonateBeanService">  
        <port name="DonateBeanPort" binding="tns:DonateBeanPortBinding">  
            <soap:address location=  
                "http://localhost:9080/DonateEJB_HTTPRouter/DonateBeanService"/>  
        </port>  
    </service>  
</definitions>
```

Per JAX-WS 2.0 specification, the annotations provide information to the Java EE 5 container to create the mapping from the Java artifact to the WSDL, and to create the bindings. The WSDL file is only created when the application is deployed to a Java EE container. In the WebSphere Application Server environment, the bindings are created during deployment.

The generation can be monitored in the server logs as shown in the following messages:

```
[1/25/10 17:50:52:921 BRT] 0000000e WASAxis2Exten I   WWS7037I: The
/DonateBeanService URL pattern was configured for the
donate.ejb.impl.DonateBean servlet located in the
DonateEJB_HTTPRouter.war web module.
```

You can find the generated WSDL (DonateBeanService.wsdl) and XSD (DonateBeanService\_schema1.xsd) files in the WebSphere Application Server profile wstemp directory in a temporary directory associated with the DonateEAR deployment:

```
<profile_directory>/wstemp/wstemp/app_<random>donate.ejb.impl.DonateBe
an
```

Under this directory you can find:

- ▶ DonateBeanService.wsdl
- ▶ DonateBeanService\_schema1.xsd
- ▶ A donate directory that contain the automatically generated jax-ws helper classes (DonateToFund, DonateToFundResponse)

---

#### *Example 11-2 Generated WSDL file*

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DonateBeanService"
targetNamespace="http://impl.ejb.donate/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://impl.ejb.donate/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://impl.ejb.donate/"
schemaLocation="DonateBeanService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="donateToFundResponse">
    <part name="parameters" element="tns:donateToFundResponse">
```

```

        </part>
    </message>
    <message name="donateToFund">
        <part name="parameters" element="tns:donateToFund">
        </part>
    </message>
    <portType name="DonateBean">
        <operation name="donateToFund">
            <input message="tns:donateToFund">
            </input>
            <output message="tns:donateToFundResponse">
            </output>
        </operation>
    </portType>
    <binding name="DonateBeanPortBinding" type="tns:DonateBean">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="donateToFund">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="DonateBeanService">
        <port name="DonateBeanPort" binding="tns:DonateBeanPortBinding">
            <soap:address
location="http://localhost:9080/DonateEJB_HTTPRouter/DonateBeanService"/>
        </port>
    </service>
</definitions>

```

---

## 11.5 Test the Web service using the Web Services Explorer

The workbench contains a generic test tool called the Web Services Explorer that enables you to load a WSDL file, either within the workspace or available anywhere over HTTP, and manually test the referenced Web service.

1. In the Services view, select **DonateEJB (...)** (under JAX-WS) and right click **Test with Web Services Explorer**.

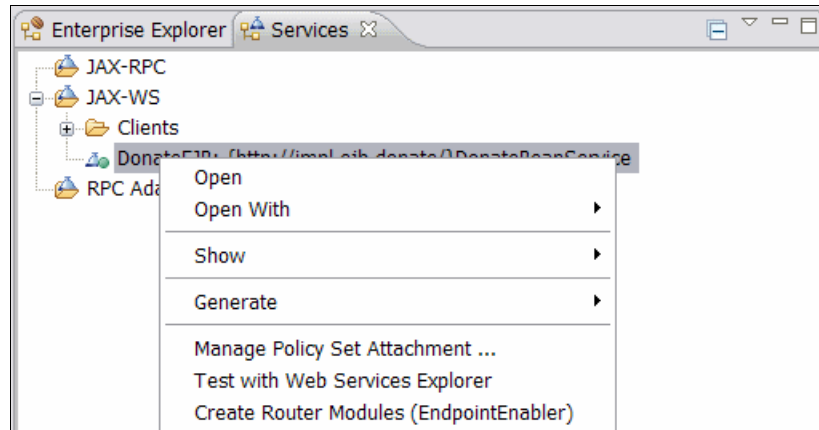


Figure 11-4 Open the Web Services Explorer

2. The Web Services Explorer explorer is loaded with the same WSDL as shown in Example 11-2 on page 424.

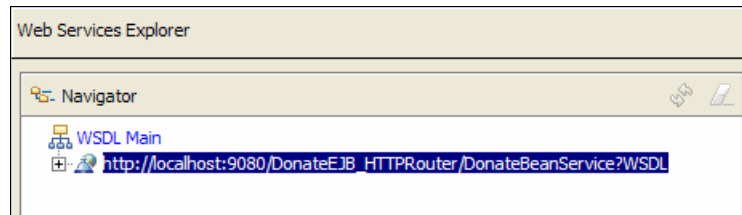


Figure 11-5 WSDL loaded in the Web Services Explorer

3. In the Navigator pane, select **WSDL Main** → **http://localhost...** → **DonateBeanService** → **DonateBeanPortBinding** → **donateToFund**.
4. In the Actions pane, which now displays Invoke a WSDL Operation:
  - a. Set arg0 (employee ID): **1**
  - b. Next to arg1 (fund name): Click **Add**, and type **DonationFund**.
  - c. Set arg2 (donation amount): **1**

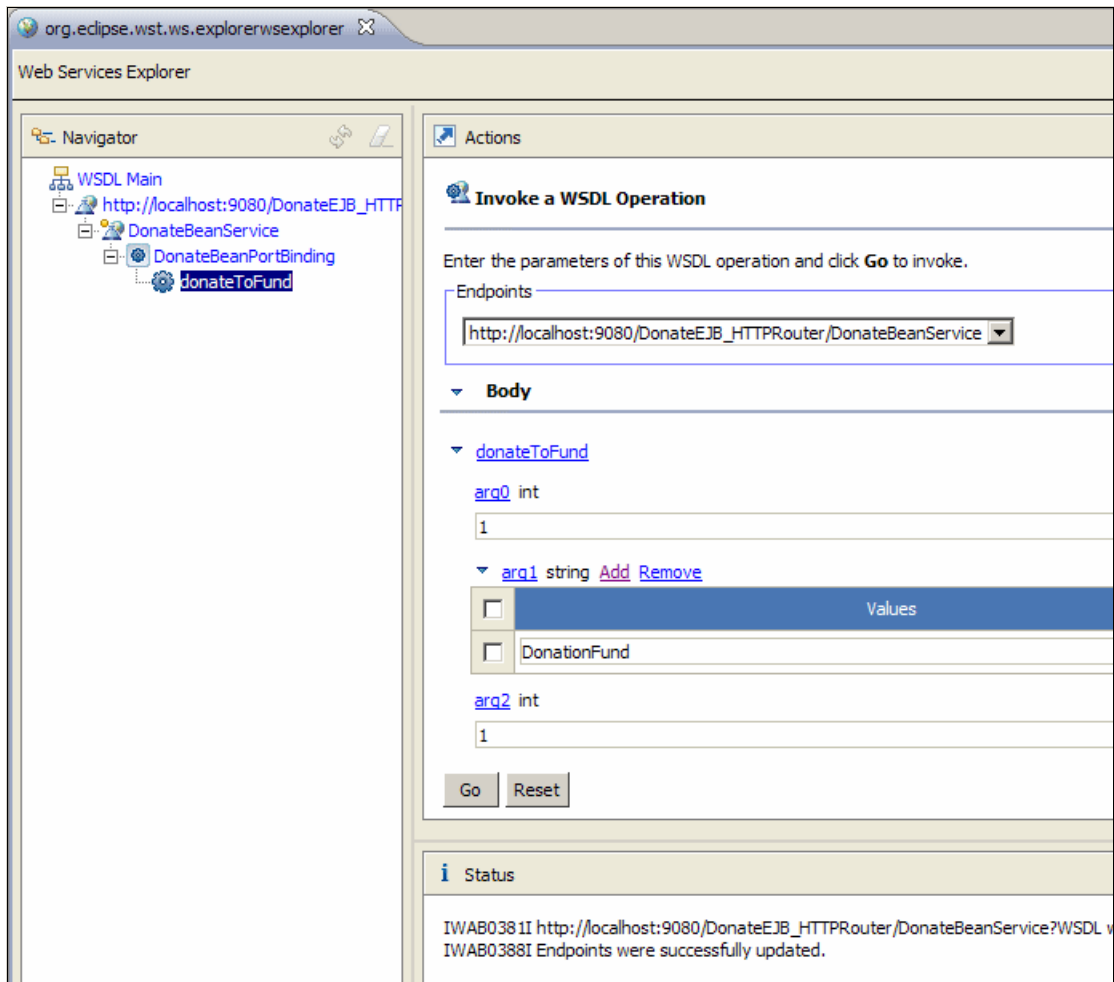


Figure 11-6 Invoke the WSDL operation

5. Click **Go**. The Web service is invoked and the result is displayed in the Status pane.

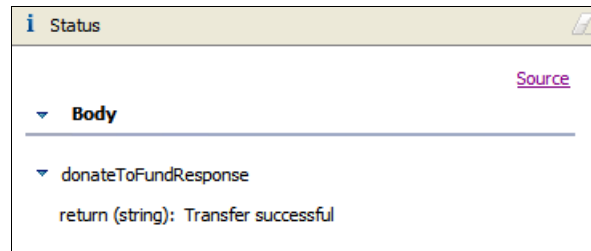


Figure 11-7 Results - success

6. In the Status pane, click **Source** to see the SOAP input and output messages:

SOAP Request Envelope:

```
<soapenv:Envelope>
  <soapenv:Body>
    <q0:donateToFund>
      <arg0>1</arg0>
      <arg1>DonationFund</arg1>
      <arg2>1</arg2>
    </q0:donateToFund>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Response Envelope:

```
<soapenv:Envelope>
  <soapenv:Body>
    <donateToFundResponse>
      <return>Transfer successful</return>
    </donateToFundResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

7. In the Actions pane, invoke the Web service with an employee ID of **21**. Note the error message in the Status pane.

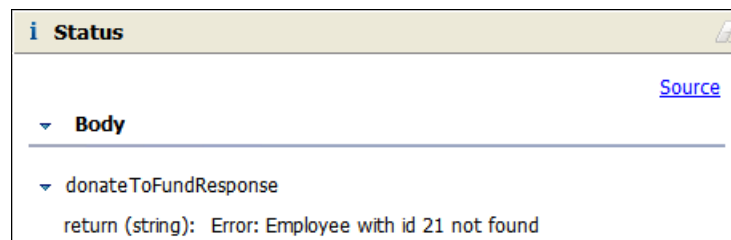


Figure 11-8 Results - with error message



8. Close the Web Services Explorer.

### Off course?

If the Web service invocation fails with the fault:

```
<soapenv:Fault
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <faultcode>soapenv:Server</faultcode>
  <faultstring>java.rmi.AccessException: ; nested exception is:
com.ibm.websphere.csi.CSIAccessException: SECJ0053E: Authorization
failed for /UNAUTHENTICATED while invoking
(Been)DonateEAR#DonateEJB.jar#DonateBean
donateToFund:int,java.lang.String,int:5 Subject: Principal:
/UNAUTHENTICATED Public Credential:
com.ibm.ws.security.auth.WSCredentialImpl@465b465b is not granted any
of the required roles: DMANAGERS_ROLE DUSERS_ROLE</faultstring>
```

Go back to 11.3, “Disable Java EE EJB declarative security” on page 414 and verify that the DonateBean was changed correctly.

## 11.6 Create and test a Web service client

The WSDL is used to allow other systems to implement Web services clients (Java EE or non-Java EE) of the published service. In this section you create and test a Web service client based on the WSDL file.

### 11.6.1 Create and test the Web service client

In this section we create a Web service client:

1. In the Services view, select **DonateEJB (...)** (under JAX-WS) and right click **Generate → Client**.
2. At the resulting Web Service Client / Web Services pop-up perform the following actions. The final configuration of this pop-up is shown in Figure 11-10 on page 431.
  - a. Verify that the Service definition is set to **http://localhost:9080/DonateWeb/DonateBeanService/DonateBeanService.wsdl**.
  - b. Verify that Client type is set to **Java Proxy**.
  - c. Use the slide on the left to set the level of the client generation to the highest level (**Test client**).

**Behind the scenes:** The slider allows to have different levels of the Web Services client generation:

Develop	generate only the client code.
Assemble	associate the client to an EAR.
Deploy	create the deployment code for the client.
Install	install the client on the selected server.
Start	start the server after the client is generated.
Test Client	generate test artifacts and start them in a browser.

- d. Under Configurations, select either **Server** or **Web service runtime**. At the resulting Client Environment Configuration pop-up:
- Set Client-Side Environment Selection to **Choose server first**.
  - Set Server to **Existing Servers** → **ExperienceJEE Test Server**.
  - Set Web service runtime to **IBM WebSphere JAX-WS**.
  - Click **OK**.

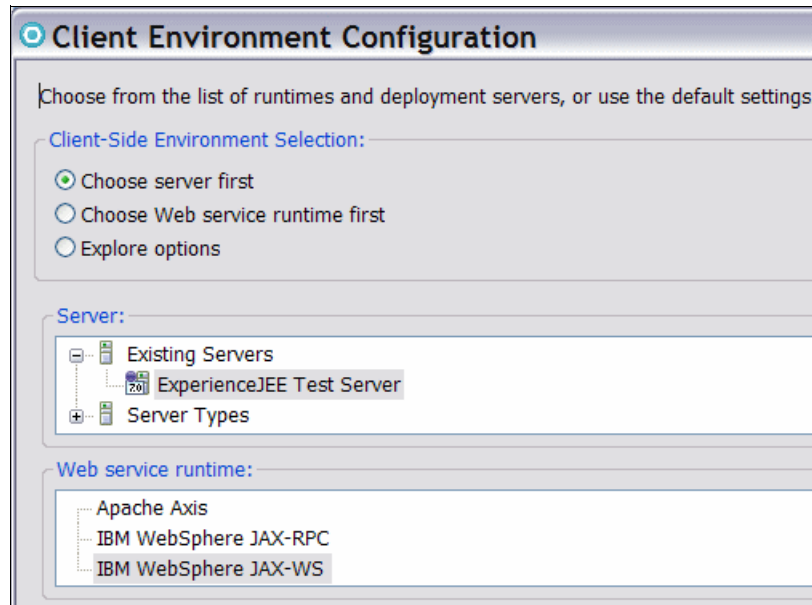


Figure 11-9 Client environment configuration

- e. Back at the Web Service Client / Web Services pop-up, select either **Client project** or **Client EAR project**. At the resulting Specific Client Project Settings pop-up:
  - i. Set Client project to **DonateWSClient**. Note that this value will not appear in the pull-down list and you will have to manually type it in.
  - ii. Set Client project type to **Dynamic Web Project**.
  - iii. Set Client EAR project to **DonateWSClientEAR**
  - iv. Click **OK**.
- f. Back at the Web Service Client / Web Services pop-up, verify that the options are set as shown in Figure 11-10 and click **Next**.

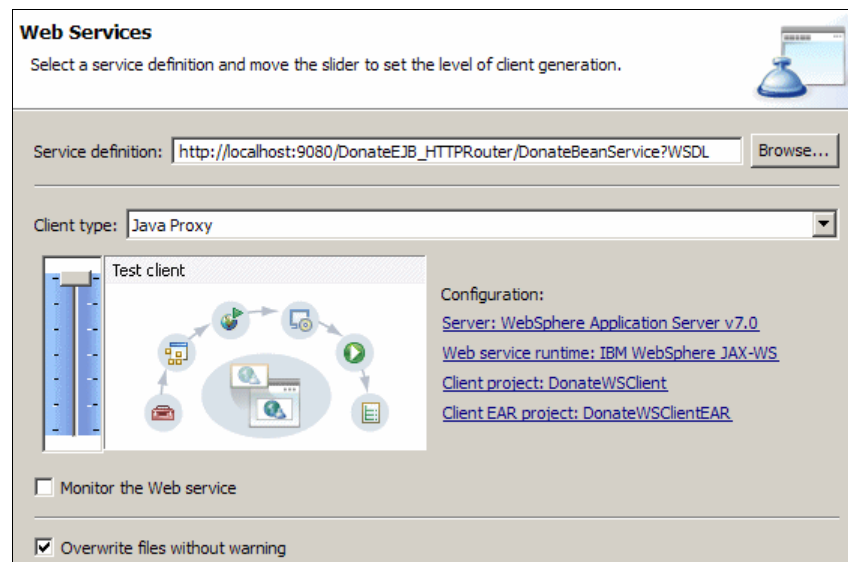


Figure 11-10 Web Service Client wizard pop-up

3. At the Web Service Client / WebSphere JAX-WS Web Service Client Configuration pop-up, accept all defaults and click **Next**.

**Behind the scenes:** this pop-up contains advanced and override options that you typically do not need to alter.

After you click Next, the wizard generates the core Web service client and deploys the EAR to the test server.

4. At the Web Service Client /Web Service Client test pop-up, click **Finish**.

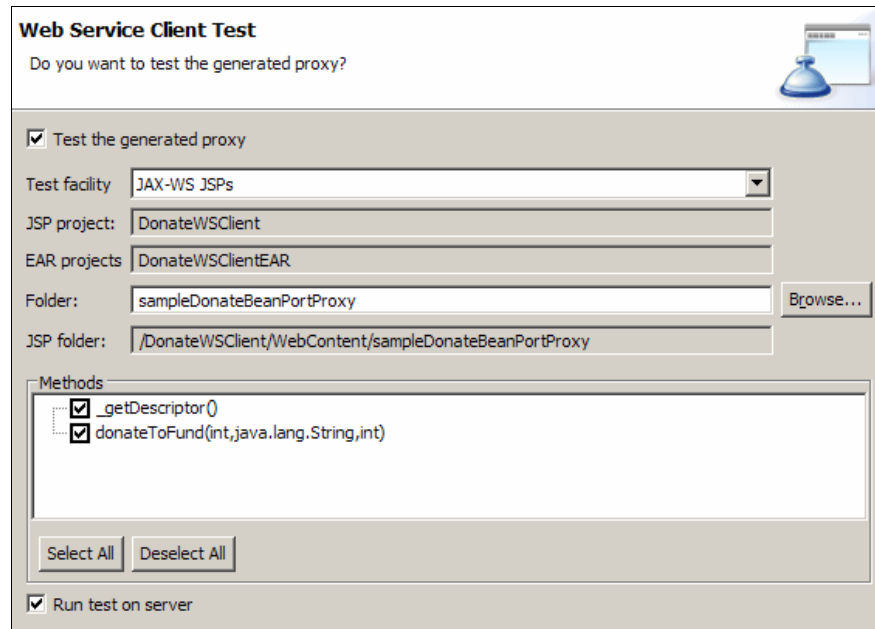


Figure 11-11 Web service client test parameters for the generated proxy

5. The WebService Test Client view is displayed. In this view you can input parameters and invoke the web service:
  - a. On the left, in the Methods pane, select **donateToFund**.
  - b. In the donateToFund pane (on the right):
    - arg0: **1**
    - arg1: **DonationFund**
    - arg2: **1**
    - Click **Invoke**.
  - c. In the Result pane, note the successful invocation.  
Transfer successful.

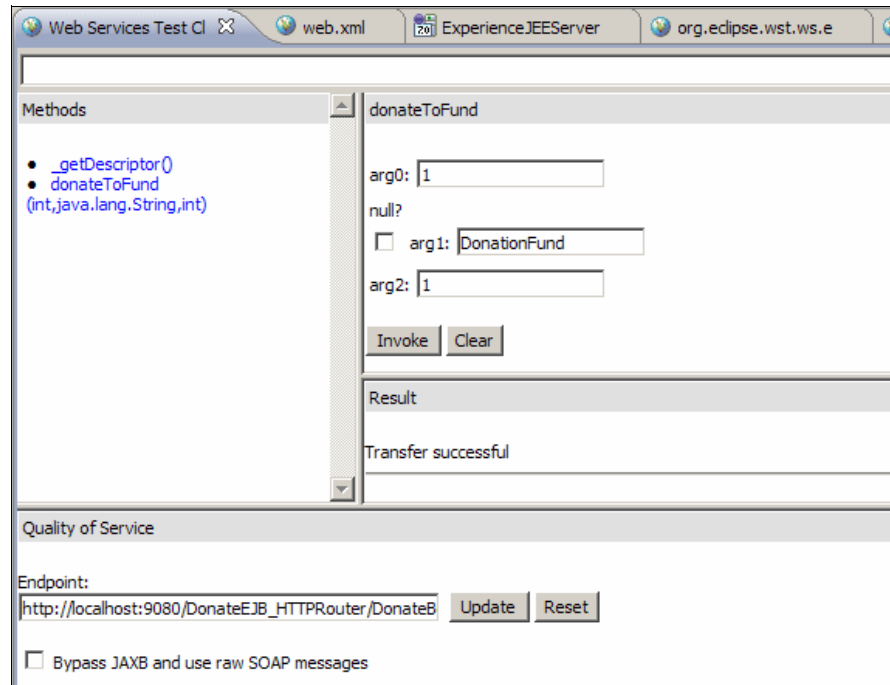
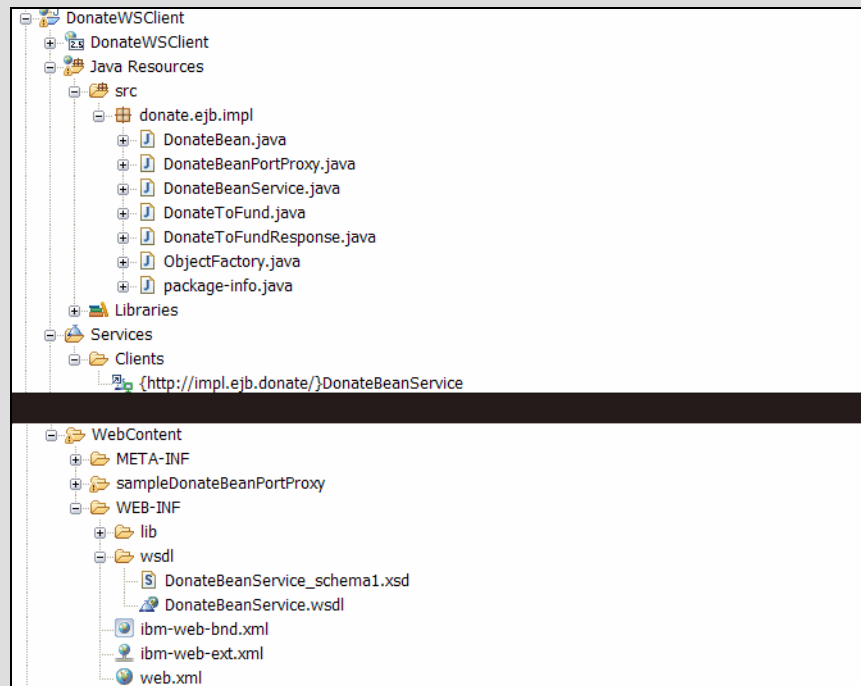


Figure 11-12 Test is invoked and runs successfully

### Behind the scenes:

The Web service client wizard uses the WSDL and schema files to generate portable artifacts that can be used by clients to access the web service:

- ▶ Service endpoint interface (SEI): `DonateBean` (refer to Example 11-3 on page 434)
- ▶ Service class: `DonateBeanService` (refer to Example 11-4 on page 435)
- ▶ Exception class mapped from `wsdl:fault` (if any)
- ▶ Async response bean derived from response `wsdl:message` (if any)
- ▶ JAXB generated value types (mapped Java classes from schema types, if any): `DonateToFund` and `DonateToFundResponse`
- ▶ Helper classes: `ObjectFactory` and `package-info`



*Example 11-3 Generated service endpoint interface (SEI): `DonateBean`*

```
@WebService(name = "DonateBean", targetNamespace =  
    "http://impl.ejb.donate/")
```

```

public interface DonateBean {
    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "donateToFund", targetNamespace =
"http://impl.ejb.donate/", className = "donate.ejb.impl.DonateToFund")
    @ResponseWrapper(localName = "donateToFundResponse", targetNamespace =
"http://impl.ejb.donate/", className =
"donate.ejb.impl.DonateToFundResponse")
    public String donateToFund(
        @WebParam(name = "arg0", targetNamespace = "") int arg0,
        @WebParam(name = "arg1", targetNamespace = "") String arg1,
        @WebParam(name = "arg2", targetNamespace = "") int arg2);
}

```

---

*Example 11-4 Generated client service class: DonateBeanService*

---

```

@WebServiceClient(name = "DonateBeanService", targetNamespace =
"http://impl.ejb.donate/", wsdlLocation =
"WEB-INF/wsdl/DonateBeanService.wsdl")
public class DonateBeanService extends Service
{
    private final static URL DONATEBEANSERVICE_WSDL_LOCATION;
    static {
        URL url = null;
        try {
            url = new URL("file:./WEB-INF/wsdl/DonateBeanService.wsdl");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        DONATEBEANSERVICE_WSDL_LOCATION = url;
    }

    public DonateBeanService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public DonateBeanService() {
        super(DONATEBEANSERVICE_WSDL_LOCATION, new
QName("http://impl.ejb.donate/", "DonateBeanService"));
    }

    @WebEndpoint(name = "DonateBeanPort")
    public DonateBean getDonateBeanPort() {
        return (DonateBean)super.getPort(new QName("http://impl.ejb.donate/",
"DonateBeanPort"), DonateBean.class);
    }
}

```

---

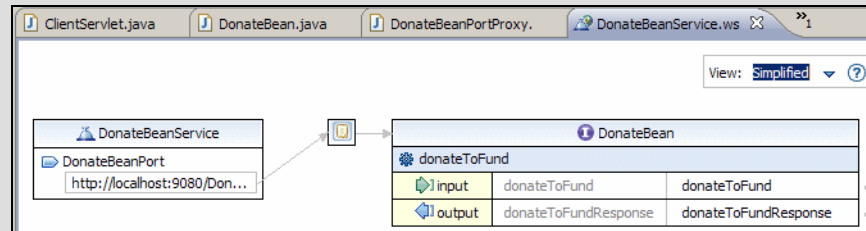
### Extra credit:

The Web Service Client wizard creates a copy of the WSDL file in the client WEB-INF/wsdl directory. The SOAP Address in this WSDL file is the address of the deployed instance of the Web service, in our case on localhost and port 9080:

```
<service name="DonateBeanService">
  <port binding="tns:DonateBeanPortBinding" name="DonateBeanPort">
    <soap:address
location="http://localhost:9080/DonateWeb/DonateBeanService"/>
  </port>
</service>
```

If you deploy the service to another server, you can change this copy of the WSDL file to change the default URL for the DonateWSClnt project, or you may simply wish to change from HTTP to HTTPS for the current deployment.

As with any WSDL file in the workbench, you can edit it using a text editor (select the WSDL, right-click and select **Open with** → Text Editor) or you can use the default WSDL editor design tab as shown below.



The workbench will open a file with the last editor that was used so if you do open a WSDL with a text editor, future edits will also open with the text editor until you explicitly reopen it with the WSDL editor.

## 11.7 Optional

The preceding steps are sufficient to perform the experiences in Chapter 12, “Implement security for the Web service” on page 447, but we only created a very basic Web service.


The **optional sections** that follow cover other Web services topics that users might encounter when developing Web services.



## 11.7.1 Monitor the SOAP traffic using TCP/IP Monitor

In this section we describe how to use the Eclipse TCP/IP Monitor feature to monitor and inspect the SOAP messages exchanged by a Web service and a Web service client.

The TCP/IP Monitor can be configured to listen for TCP/IP packages on one port and forward them to a different host and port. Here we configure it to listen to the localhost port 9081 and forward the requests to localhost port 9080. We then use the Web Services Explorer to make requests to the DonateBean Web service using the port 9081 and we are able to monitor the messages exchanged by the server and the Web Services Explorer:

1. In the workbench action bar, select **Window** → **Show View** → **Other**.
2. At the Show view pop-up, select **Debug** → **TCP/IP Monitor** and click **OK**.
3. In the resulting TCP/IP Monitor view in the lower right, click the **Menu** icon  on the top right and select **Properties**.

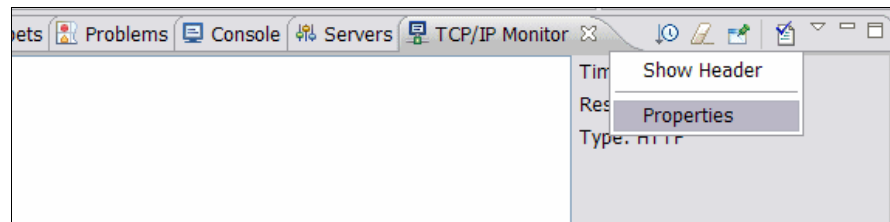


Figure 11-13 TCP/IP Monitor - select Properties

4. At the Preferences pop-up, click **Add**.
5. At the New Monitor pop-up set this values:
  - Local monitoring port: **9081**
  - Host name: **localhost**
  - Port: **9080**
  - Type: **HTTP**
  - Time-out (in milliseconds): **0**
  - a. Click **OK**.
6. Back at the Preferences popup, select the new entry, and click **Start**.
  - a. Ensure that the status changes from Stopped to Started, and then click **OK** to close the pop-up.

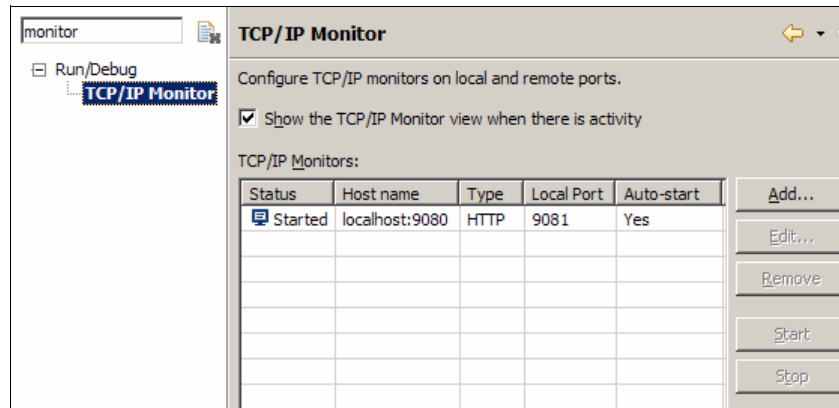


Figure 11-14 Start the monitor

### Behind the scenes:

You can also configure the TCP/IP Monitor by selecting **Window** → **Preferences**, and at the Preferences pop-up select **Run/Debug** → **TCP/IP Monitor**.

7. In the Services view (on the left, next to the Enterprise Explorer tab, select **DonateEJB(...)** (under JAX-WS) and right click on **Test with Web Services Explorer**.
8. In the WSDL Binding Details you get the list of endpoints of the WSDL. Note that a new endpoint entry using port 9081 was added to the list.
  - a. Select the endpoint using port 9081 and click **Go**

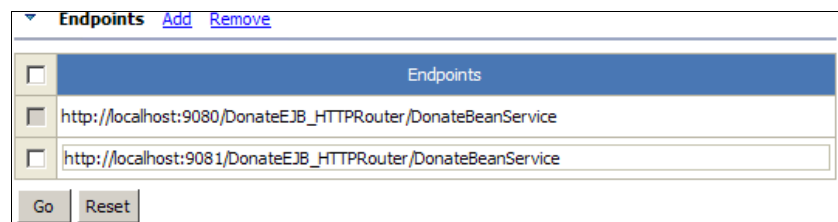


Figure 11-15 Select the endpoint

9. In the Navigator pane, select the **donateToFund** operation.

10. In the Actions pane, which now displays Invoke a WSDL Operation. Set this value arg0 (employee ID): **1**.
  - a. Next to arg1 (fund name): Click **Add**, and type **DonationFund**.
  - b. Set arg2 (donation amount): **1**.
  - c. Click **Go**.
11. Select the TCP/IP Monitor view and note that a request is shown.
  - a. Change the message representation from **byte** to **XML** to see the formatted display of the SOAP request and response.

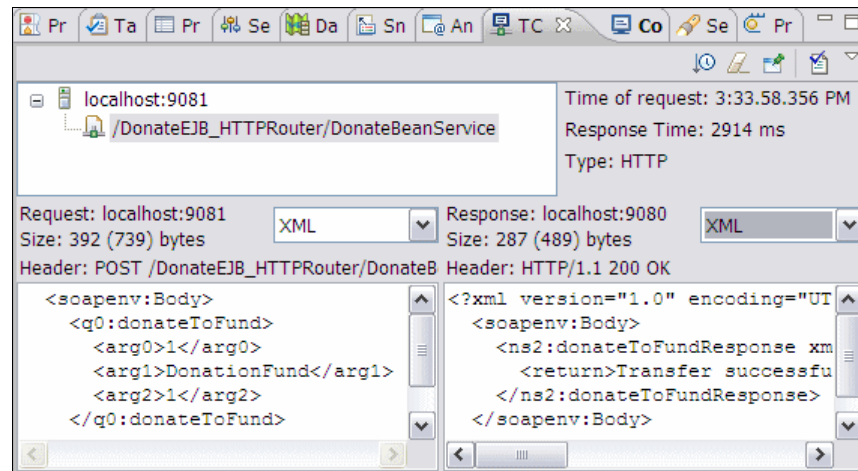


Figure 11-16 TCP/IP Monitor results

### Behind the scenes:

On the left we can see the SOAP message sent to the server with the service request. In it we can see the SOAP message body having the operation requested (donateToFund) and the arguments values (Example 11-5).

#### *Example 11-5 SOAP request envelope*

---

```
<soapenv:Envelope xmlns:q0="http://impl.ejb.donate/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns2:donateToFund xmlns:ns2="http://impl.ejb.donate/">
      <arg0>1</arg0>
      <arg1>DonationFund</arg1>
      <arg2>1</arg2>
    </ns2:donateToFund>
  </soapenv:Body>
</soapenv:Envelope>
```

---

On the right we can see the SOAP message returned with the results of the Web service execution (Example 11-6).

#### *Example 11-6 SOAP response envelope*

---

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns2:donateToFundResponse xmlns:ns2="http://impl.ejb.donate/">
      <return>Transfer successful</return>
    </ns2:donateToFundResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

---

The TCP/IP Monitor can be used to monitor any interactions between a Web service and its client. To use the TCP/IP Monitor, just change the service endpoint URL to point to the monitored port.

## 11.7.2 Create a client servlet

In this section we create a servlet that uses the `@WebServiceRef` annotation to use the artifacts generated by the Web Services Client wizard. It allow us to develop a client code that handles the Web Service response and manipulates it.

To use the `@WebServiceRef` annotation we need to have our class managed by the Java EE Environment. In this case we are creating a servlet which is a managed bean.

1. In the Enterprise Explorer, select **DonateWSClientWeb**, right-click and select **New** → **Servlet**.
2. At the Create Servlet pop-up set these values:
  - Java package name: **donate.client**
  - Class name: **ClientServlet**
- a. Click **Finish**.

### Behind the scenes:

A new servlet is created and mapped to a `/ClientServlet` URL. The wizard creates the `HttpServlet` Java class in the project. It also creates the definition of the servlet and the following mapping in the `web.xml` deployment descriptor:

```
<servlet>
  <description></description>
  <display-name>ClientServlet</display-name>
  <servlet-name>ClientServlet</servlet-name>
  <servlet-class>donate.client.ClientServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ClientServlet</servlet-name>
  <url-pattern>/ClientServlet</url-pattern>
</servlet-mapping>
```

- b. At the resulting `ClientServlet.java` editor, insert the following lines after the class definition:

```
public class ClientServlet extends HttpServlet{

    @WebServiceRef
    DonateBeanService service;

    @WebServiceRef(value=DonateBeanService.class)
    DonateBean mybean;
```

### Behind the scenes:

Referencing a Web service was been simplified in Java EE 5. The annotation used to reference the service is:

**@WebServiceRef** The `javax.xml.ws.WebServiceRef` annotation is used to declare a reference to a Web service. It can be used to reference a generated service class or a generated service endpoint interface (in that case the generated service class type parameter must be always present).

In the client servlet we use both references. First we use a reference to the generated service class (`DonateBeanService`), and the second reference binds to a generated service endpoint interface (`DonateBean`).

Note that each technique works on its own.

3. In the `doGet` method, delete the method body, and insert the **F11.1 Web Service Client Servlet** snippet from the Snippets view (Example 11-7).

*Example 11-7 F11.1 Web Service Client Servlet*

---

```
protected void doGet(.....) throws .... {
    PrintWriter out = response.getWriter();
    response.setContentType("text/plain");
    out.println("Calling DonateBean Web service... ");
    try {
        DonateBean bean = service.getDonateBeanPort();
        out.print("Response using service class: ");
        out.println(bean.donateToFund(1, "DonationFund", 1));
        out.print("Response using SEI directly: ");
        out.println(mybean.donateToFund(1, "DonationFund", 1));
    } catch (Exception e) {
        out.println("EXCEPTION: " + e.getLocalizedMessage());
    }
}
```

---

4. Replace the `doPost` method body with `doGet(request, response);`.

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
}
```

5. Organize imports (**Ctrl+Shift+O**) to resolve:

```
import java.io.PrintWriter
import donate.ejb.impl.DonateBean;
```

```
import donate.ejb.impl.DonateBeanService;
import javax.xml.ws.WebServiceRef;
```

6. Save and close `ClientServlet.java`.

### Behind the scenes:

There are different ways of referencing a Web Service in a servlet:

1. Inject the service class.

Using the service class we have to get the service proxy, also known as a port. The method to get a service proxy is usually named `get<proxyname>Port`. Here are the lines to retrieve the service proxy for the `DonateBean` Web service and invoke the method:

```
DonateBean bean = service.getDonateBeanPort();
bean.donateToFund(1, "DonationFund", 1)
```

2. Provide the value argument that contains the service class type.

In this case, the container injects a SEI by instantiating the service class and retrieving the port. We can call the method directly:

```
mybean.donateToFund(1, "DonationFund", 1)
```

The advantage of using the second reference is that the code is reduced, and we do not have to call the method to return the Web service port. However, in some cases the service class might be customized and the method name to return the Web service port might have a different name.

## 11.7.3 Run the client servlet

Now we can run the servlet by publishing the application:

1. Publish the `DonateEAR` enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**)
2. In the Enterprise Explorer, select **ClientServlet** (under `DonateWSCClient/DonateWSCClient/Servlet`), right-click and select **Run-as** → **Run on server**.

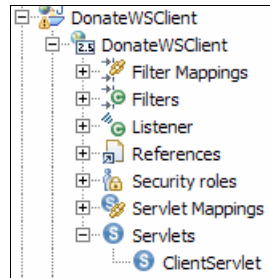


Figure 11-17 ClientServlet

3. If you see the Run On Server pop-up, select the **ExperienceJEE Test Server** and click **Finish**.
4. The result of the Web service client is displayed in the browser window.

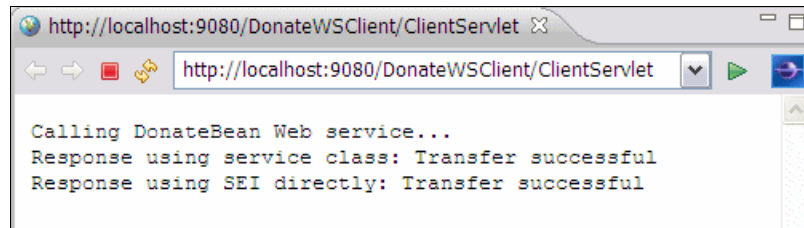


Figure 11-18 Web service client results

## 11.8 Explore!

JAX-WS specification defines other technologies to improve Web services performance.

### 11.8.1 Handlers in JAX-WS

JAX-WS applies the chain of responsibility pattern by providing interceptors to its architecture. These interceptors are called *handlers* in JAX-WS, and are used to do additional processing of inbound and outbound messages.

JAX-WS defines two types of handlers:

SOAP handlers	Used to process SOAP-specific information such as SOAP security headers.
---------------	--------------------------------------------------------------------------



Logical handlers      Used when there is no need to access the SOAP message. For example, logical handlers can be used for message validation or to implement REST style Web services.

Here are some references about JAX-WS handlers:

- ▶ A little bit about handlers in JAX-WS:  
[http://jax-ws.dev.java.net/articles/handlers\\_introduction.html](http://jax-ws.dev.java.net/articles/handlers_introduction.html)
- ▶ Writing a handler in JAX-WS:  
<http://www.java-tips.org/java-ee-tips/java-api-for-xml-web-services/writing-a-handler-in-jax-ws.html>

## 11.8.2 Asynchronous Web service invocation

JAX-WS supports that a Web service client can interact with a Web services in a non-blocking, asynchronous approach. This is done by adding support for both a polling and callback mechanism when calling Web services asynchronously:

- ▶ Using a *polling* model, a client can issue a request, get a response object back, which is polled to determine if the server has responded. When the server responds, the actual response is retrieved.
- ▶ Using the *callback* model, the client provides a callback handler to accept and process the inbound response object.

Both the polling and callback models enable the client to focus on continuing to process work without waiting for a response to return, while providing for a more dynamic and efficient model to invoke Web services.

For more information about asynchronous Web service invocation, refer to the following Web sites:

- ▶ Asynchronous Web service invocation with JAX-WS 2.0  
<http://today.java.net/pub/a/today/2006/09/19/asynchronous-jax-ws-web-services.html?page=last>
- ▶ Developing asynchronous Web services with Axis2:  
<http://www.ibm.com/developerworks/library/ws-axis2/index.html>

## 11.8.3 Attachments performance improvement

SOAP Message Transmission Optimization Mechanism (MTOM) is a standard that was developed by the World Wide Web Consortium (W3C). MTOM describes a mechanism for optimizing the transmission or wire format of a SOAP message by selectively re-encoding portions of the message while still presenting an XML Information Set (Infoset) to the SOAP application.

MTOM uses the XML-binary Optimized Packaging (XOP) in the context of SOAP and MIME over HTTP. XOP defines a serialization mechanism for the XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but to any XML Infoset and any packaging mechanism. It is an alternate serialization of XML that just happens to look like a MIME multipart or related package, with XML documents as the root part.

That root part is very similar to the XML serialization of the document, except that base64-encoded data is replaced by a reference to one of the MIME parts, which is not base64 encoded. This reference enables you to avoid the bulk and overhead in processing that is associated with encoding. Encoding is the only way binary data can work directly with XML.

For more information consult:

- ▶ Rational Application Developer Information Center *SOAP MTOM*:  
<http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.webservice.wsf.doc/topics/cmtom.html>
- ▶ The Official MTOM specification:  
<http://www.w3.org/TR/soap12-mtom/>
- ▶ Apache *MTOM Guide -Sending Binary Data with SOAP*:  
[http://ws.apache.org/axis2/1\\_0/mtom-guide.html](http://ws.apache.org/axis2/1_0/mtom-guide.html)

### 11.8.4 Creating a top-down Web service from a WSDL

When creating a Web service using a top-down approach, first you design the implementation of the Web service by creating a WSDL file. You can do this using the WSDL editor. You can then use the Web Service wizard to create the Web service and skeleton Java classes to which you can add the required code.

For more information consult:

- ▶ Redbook: Rational Application Developer V7.5 Programming Guide, Chapter 18. Developing Web services applications  
<http://www.redbooks.ibm.com/redpieces/abstracts/sg247672.html>



# Implement security for the Web service

In this chapter, we update the Web service and Web service client from the previous chapter to implement authentication.

## 12.1 Learn!

Figure 12-1 shows implementing authentication.

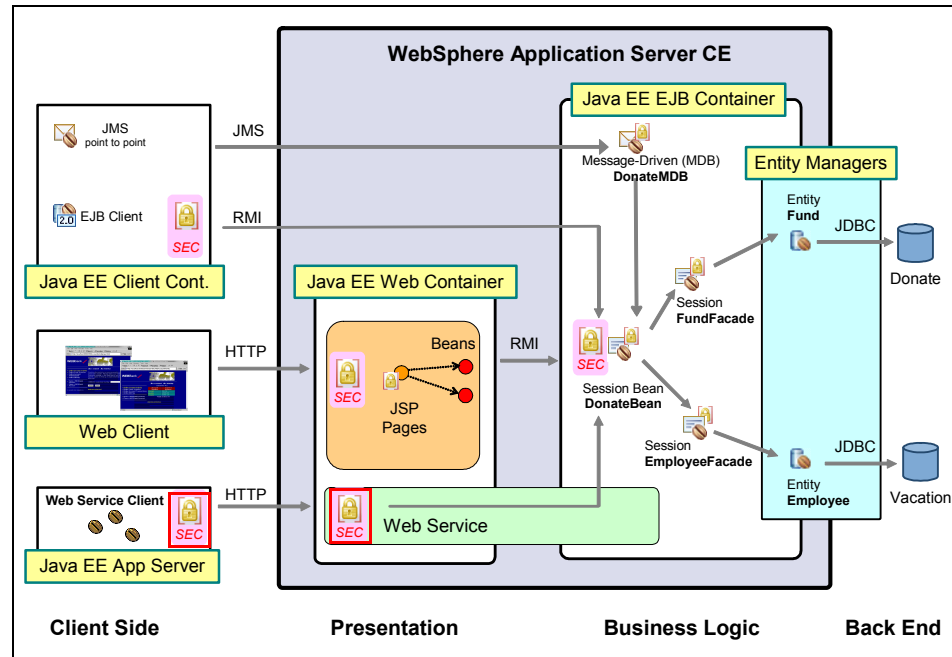


Figure 12-1 Donate application: Web service security

The JAX-WS specification requires that its implementations must support transport level security (for example HTTPS). This way the channel is protected between two parties. It can authenticate user using HTTP basic authentication (Figure 12-2 on page 449) that is a method to allow a client to provide credentials to a server when executing a request. It assumes that the connection between the server and the client is secure. So it is important to have an SSL/HTTPS connection when using this authentication method. JAX-WS defines the annotations that can be used to protect the method based on roles.

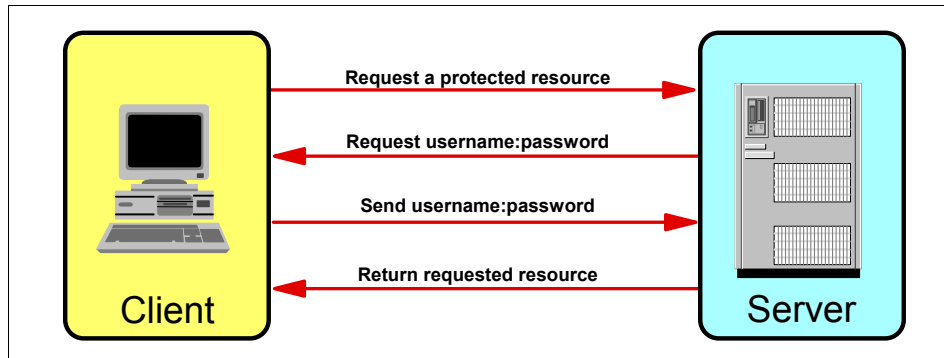


Figure 12-2 HTTP Basic Authentication schema

The JAX-WS specification does not specify the entire WS-\* security protocols. However in certain scenarios the service consumer and service provider are separated by intermediaries (e.g. an Enterprise Service Bus). The WS-Security standard describes how to add confidentiality, integrity and authentication to a SOAP message (Figure 12-3). These technologies can provide message-level security that is an end-to-end security solution. Using message-level security the message is encrypted using the public key of the final destination, this way only the receiver can decrypt the message. The message can be stored anywhere and delivered asynchronously without any threat to exposing the message content.

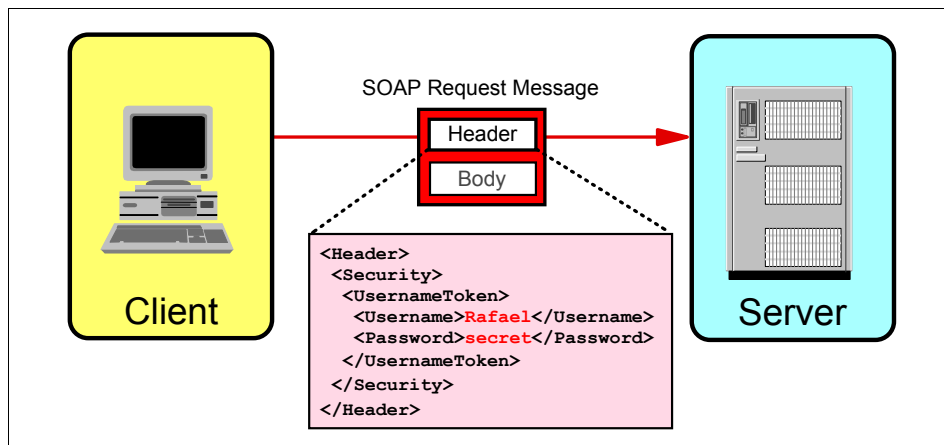


Figure 12-3 Security in the SOAP header

Web services security for WebSphere Application Server v7.0 is based on standards included in the Organization for the Advancement of Structured Information Standards (OASIS) Web services security (WSS) Version 1.0/1.1

specification, the Username Token Profile 1.0/1.1, and the X.509 Certificate Token Profile 1.0/1.1.

## **Authentication**

Authentication is used to ensure that parties within a business transaction are really who they claim to be; thus proof of identity is required. This proof can be claimed in various ways:

- ▶ One simple way is by presenting a user identifier and a password. This is referred to as a username token in WS-Security domain
- ▶ A more complex way is to use an X.509 certificate issued by a trusted certificate authority.

The certificate contains identity credentials and has a pair of private and public keys associated with it. The proof of identity presented by a party includes the certificate itself and a separate piece of information that is digitally signed using the certificate's private key. By validating the signed information using the public key associated with the party's certificate, the receiver can authenticate the sender as being the owner of the certificate, thereby validating their identity.

Two WS-Security specifications, the Username Token Profile 1.0/1.1 and the X.509 Certificate Token Profile 1.0/1.1, describe how to use these authentication mechanisms with WS-Security.

## **Message integrity**

To validate a message has not been tampered with or corrupted during its transmission over the Internet, the message can be digitally signed using security keys. The sender uses the private key of their X.509 certificate to digitally sign the SOAP request. The receiver uses the sender's public key to check the signature and identity of the signer. The receiver signs the response with their private key, and the sender is able to validate the response has not been tampered with or corrupted using the receiver's public key to check the signature and identity of the responder. The WS-Security: SOAP Message Security 1.0/1.1 specification describes enhancements to SOAP messaging to provide message integrity.

## **Message confidentiality**

To keep the message safe from eavesdropping, encryption technology is used to scramble the information in Web services requests and responses. The encryption ensures that no-one accesses the data in transit, in memory, or after it has been persisted, unless they have the private key of the recipient. The WS-Security: SOAP Message Security 1.0/1.1 specification describes enhancements to SOAP messaging to provide message confidentiality.

There are two options to configure WS-Security for JAX-WS Web services:

- Policy set Programming API for securing SOAP message with Web Service Security (WSS API)
- Service Programming Interfaces (SPI) for a service provider.

We will use Policy set in our examples.

## **Policy set**

You can use policy sets to simplify configuring the qualities of service for Web services and clients. Policy sets are assertions about how Web services are defined. Using policy sets, you can combine configurations for different policies. You can use policy sets with JAX-WS applications, but not with JAX-RPC applications.

A policy set is identified by a unique name. An instance of a policy set consists of a collection of policy types. An empty policy set has no policy instance defined. Policies are defined on the basis of a quality of service. Policy definitions are typically based on WS-Policy standards language. For example, the WS-Security policy is based on the current WS-SecurityPolicy language from the Organization for the Advancement of Structured Information Standards (OASIS) standards. Policy sets omit application or user-specific information, such as keys for signing, key store information, or persistent store information. Instead, application and user-specific information is defined in the bindings. Typically, bindings are specific to the application or the user, and bindings are not normally shared. On the server side, if you do not specify a binding for a policy set, a default binding will be used for that policy set. On the client side, you must specify a binding for each policy set.

A policy set attachment defines which policy set is attached to service resources, and which bindings are used for the attachment. The bindings define how the policy set is attached to the resources. An attachment is defined outside of the policy set, as metadata associated with the application. To enable a policy set to work with an application, a binding is required.

Policy sets does not contain environment specific settings like key stores or passwords. These types of settings are configured in bindings.

## **WebSphere 7.0 support for Policy Sets**

WebSphere Application Server v7 comes with 18 pre-configured production level policy sets that can be used and 4 sample bindings. Rational Application Developer 7.5 also comes prepackaged with a set of policy. After importing it the Rational Application Developer can attach the policy sets and policy set bindings to the service provider.

## 12.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed all the applications of Part 2, “Core Java EE application” on page 121. In addition, you must have implemented the Web services in Chapter 11, “Create the Web service” on page 409.

## 12.3 Re-enable Java EE EJB declarative security



In this section we perform the set of tasks required to re-enable the Java EE EJB declarative security that we disabled in 11.3, “Disable Java EE EJB declarative security” on page 414:

1. Re-enable declarative security in the DonateBean session EJB:




- In the Enterprise Explorer, open **DonateBean** (in DonateEJB/ejbModule/donate.ejb.impl), and switch the security annotations for both donateToFund methods (remove the comment marker from @RolesAllowed and add a comment marker to @PermitAll):

```
@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
//@PermitAll
public void donateToFund(Employee employee, Fund fund, int hours)
.....
```

```
@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
//@PermitAll
@WebMethod
public String donateToFund(int employeeId, String fundName, ...)
.....
```

- Organize the imports (**Ctrl+Shift+O**) to resolve javax.annotation.security.RolesAllowed-
  - Save and close DonateBean.java.
2. Ensure that the test environment is running:
    - If the Derby Network Server is not running, open a command window, change directory to:
      -  C:\IBM\SDP\runtimes\base\_v7\derby\bin\networkServer
      -  /opt/IBM/SDP/runtimes/base\_v7/derby/bin/networkServer
    - and execute:



-  `startNetworkServer.bat`
  -  `startNetworkServers.sh`
- If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
- If the server instance starts successfully, you will find a message **Server started** in the Console view.
3. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).

## 12.4 Test the Web service as-is

In this section we execute the Web Service Client created in the previous chapter pointing to the current Web service artifacts to verify that the `donateToFund` method fails with an authorization exception.

1. In the Java EE perspective, select **TestClient.jsp** (in `DonateWSCClient/WebContent/sampleDonateBeanPortProxy`) and right click **Run As** → **Run on Server**.
2. In the resulting Web Service Test Client page, under the Methods pane (on the left) select **donateToFund**.
3. In the resulting `donateToFund` pane (on the right):
  - a. Set these parameter values:
    - `arg0:1`
    - `arg1: DonationFund`
    - `arg2: 1`
  - b. Click **Invoke**.
4. In the Result pane, note the unsuccessful invocation:
 

```
Exception: javax.xml.ws.soap.SOAPFaultException:
security.wssecurity.WSSContextImpl.s02:
com.ibm.websphere.security.WSSecurityException: Exception
org.apache.axis2.AxisFault: CWSS6500E: There is no caller identity
candidate that can be used to login. occurred while running action:
com.ibm.ws.wssecurity.handler.WSSecurityConsumerHandler$1@21932193 Message:
security.wssecurity.WSSContextImpl.s02:
com.ibm.websphere.security.WSSecurityException: Exception
org.apache.axis2.AxisFault: CWSS6500E: There is no caller identity
candidate that can be used to login. occurred while running action:
com.ibm.ws.wssecurity.handler.WSSecurityConsumerHandler$1@21932193
```

### Extra credit:

Using the TCP/IP monitor you can inspect the SOAP XML message returned from the server:

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <soapenv:Fault>
      <faultcode>soapenv:Server</faultcode>
      <faultstring>security.wssecurity.WSSContextImpl.s02:
com.ibm.websphere.security.WSSecurityException: Exception
org.apache.axis2.AxisFault: CWSS6500E: There is no caller
identity candidate that can be used to login. occurred while
running action:
com.ibm.ws.wssecurity.handler.WSSecurityConsumerHandler$1@21932193
</faultstring>
      <detail/>
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

## 12.5 Update the Web service to support secure access

In this section we secure the EJB Web service with authentication using an end-to-end protection based on message level security. We are going to use the Username WSSecurity.

### 12.5.1 Create the policy set

In order to create the policy set we are going to customize an existing policy set in WebSphere Application Server and export it to be used in the workbench. The administrative console is used to perform this task.

Although the WebSphere Application Server already has policy sets configured, we are creating a new policy set that will contain only the specific policy we are interested in: WS-Security. This way we can clearly demonstrate the security operation, avoiding several message encryption, message signing, and additional headers such as WS-Addressing.

1. Start the Admin Console using the same steps as describe in 4.6.1, "Start the Administrative Console" on page 100.

- a. Recall that in the Admin Console login screen you use the javaeeadmin user ID and password that you defined in 3.4, “Create the ExperienceJEE server profile” on page 62.
2. In the Admin Console, on the left select **Services** → **Policy sets** → **Application policy sets**.
3. On the right, in the Application policy sets panel, enable the check box next to **Username WSSecurity default** and click **Copy**.

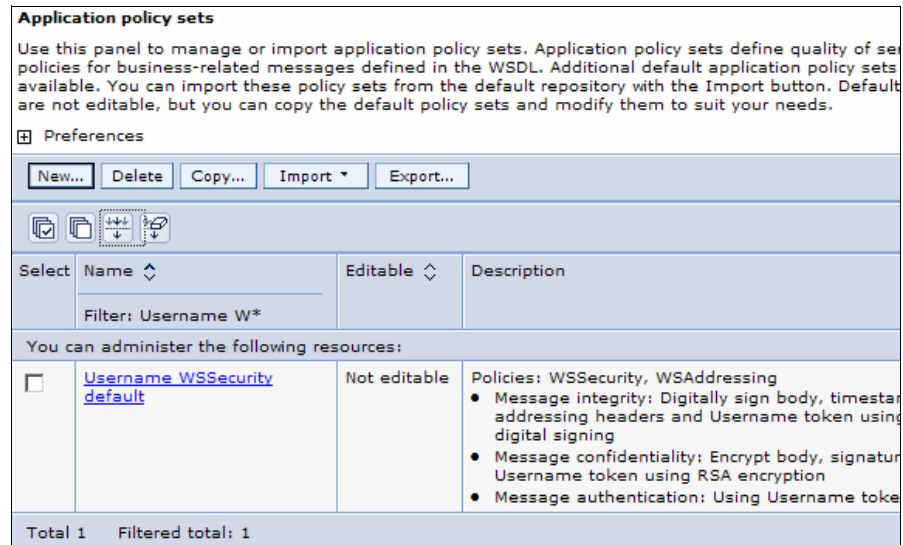


Figure 12-4 Select and copy the policy

#### Behind the scenes:

The standard list of application policy set is much longer -- this specific screen shot used a filter to show only the desired entry using the filter icon [⊞] (beneath the New/Delete/Copy/Import/Export command options).

4. In the resulting Application policy sets → Copy... panel, set Name to **UsernameMinimum** and click **OK**.
5. Back in the Application policy sets panel, click on the new **UsernameMinimum** entry.
6. In the Application policy sets → UsernameMinimum panel, enable the check box next to **WS-Addressing** and click **Delete**.

Add ▾ Delete Enable Disable			
Select	Policy ▾	State ▾	Description
You can administer the following resources:			
<input checked="" type="checkbox"/>	<a href="#">WS-Addressing</a>	Enabled	Policies for addressing Web services using endpoint references and message addressing properties.
<input type="checkbox"/>	<a href="#">WS-Security</a>	Enabled	Policies for sending security tokens and providing message confidentiality and integrity, based on the OASIS Web Service Security and Token Profiles specifications.
Total 2			

Figure 12-5 Delete the WS-Addressing policy

### Behind the scenes:

WS-addressing is a Web service specification that takes information that is typically sent in the HTTP headers (such as the implied replyTo address) and inserts this into a SOAP header. This enables extended Web service capabilities such as asynchronous replies and alternate transport mechanisms such as JMS.

This chapter focuses strictly on WS-Security; therefore, WS-Addressing is removed.

7. In the Application policy sets → UsernameMinimum panel, click the **WS-Security** entry.
8. In the Application policy sets → UsernameMinimum → WS-Security panel, click **Main policy**.
9. In the Application policy sets → UsernameMinimum → WS-Security → Main Policy panel:
  - a. Under Message Part Protection, click **Request message part protection**.
  - b. In the Application policy sets → UsernameMinimum → WS-Security → Main Policy → Request Message part protection panel:
    - i. Under Confidentiality protection → Encrypted parts, select **app\_encparts** and click **Delete**.
    - ii. Under Integrity protection → Signed parts, select **app\_signparts** and click **Delete**.
    - iii. Click **Done**.

[Application policy sets](#) > [Fred](#) > [WS-Security](#) > [Main policy](#) > [Request message part protection](#)

Message part protection policies define parts to be protected and the nature of protection.

**Confidentiality protection**

Encrypted parts	
app_encparts	<input type="button" value="Add"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>

**Integrity protection**

Signed parts	
app_signparts	<input type="button" value="Add"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>

Figure 12-6 Update the request message part protection settings

**Behind the scenes:** these setting sign and encrypt the various elements within the SOAP message, including the WS-Security header. These setting should be viewed as mandatory for a product Web services implementations. However, they are deleted in the context of this book to allow you to view the WS-Security SOAP header via the TCP/IP Monitor.

10. Back at the Application policy sets → UsernameMinimum → WS-Security → Main Policy panel:

a. Clear the **Message level protection** entry..

**Behind the scenes:** this setting indicates whether the message must be signed and encrypted. This setting should be viewed as mandatory for a product Web services implementations. However, is disabled in the context of this book to allow you to view the WS-Security SOAP header via the TCP/IP Monitor.

b. Click **Apply** (at the bottom) to save the changes.

c. Click **Save** (in the message panel on the top of the page) to commit the changes.

11. Export the policy set:

a. In the Admin Console, on the left select **Services** → **Policy sets** → **Application policy sets**.

b. On the right, in the Application policy sets panel, enable the checkbox next to **UsernameMinimum** and click **Export...** button

- c. In the Application policy sets → Export policy set archive file panel, click the **UsernameMinimum.zip** link and save in your local file system.

#### Behind the scene:

The exported file contains in the folder PolicySets / UsernameMinimum / PolicyTypes / WSSecurity the policy.xml file. This file contains the configuration of the application policy set we have configured.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
...
<sp:SupportingTokens>
  <wsp:Policy wsu:Id="request:token_auth">
    <sp:UsernameToken
      sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient">
      <wsp:Policy>
        <sp:WssUsernameToken10/>
      </wsp:Policy>
    </sp:UsernameToken>
  </wsp:Policy>
</sp:SupportingTokens>
</wsp:Policy>
```

It defines the type of policy set used, in this case the UsernameToken.

## 12.5.2 Create the client policy set bindings

The policy set does not have information about username and password. It must be configured in a policy set bindings. You must specify a valid username and password in your environment.

1. In the Admin Console, on the left select **Services** → **Policy sets** → **General client policy set bindings**.
2. On the right in the General client policy set bindings panel, enable the check box next to **Client Sample** and click **Copy**.
3. On the right in the General client policy set bindings → Copy of Client sample panel, set Name to **expjeeClient** and click **OK**.
4. Back in the General client policy set bindings panel, click on the new **expjeeClient** entry.
5. In the General client policy set bindings → expjeeClient panel, enable the check box next to all sections except for WS-Security, and click **Delete**.

Add ▾ Delete	
Select	Policy ▾
You can administer the following resources:	
<input checked="" type="checkbox"/>	<a href="#">HTTP transport</a>
<input checked="" type="checkbox"/>	<a href="#">JMS transport</a>
<input checked="" type="checkbox"/>	<a href="#">SSL transport</a>
<input checked="" type="checkbox"/>	<a href="#">WS-ReliableMessaging</a>
<input type="checkbox"/>	<a href="#">WS-Security</a>

Figure 12-7 Update the general client policy set bindings for expjeeClient

6. In the General client policy set bindings → expjeeClient panel click the **WS-Security** entry.
7. In the General client policy set bindings → expjeeClient → WS-Security panel, click **Authentication and Protection**.
8. In the General client policy set bindings → expjeeClient → WS-Security → Authentication and Protection panel, under the Authentication tokens list click the **gen\_signunametoken** entry.
9. In the General client policy set bindings → expjeeClient → WS-Security → Authentication and Protection → gen\_signunametoken panel, at the bottom under Additional Bindings click **Callback handler**.
10. In the General client policy set bindings → expjeeClient → WS-Security → Authentication → gen\_signunametoken → Callback handler panel:
  - a. Under Basic Authentication, enter the **DNARMSTRONG** username and password.
  - b. Click **OK**.

Basic Authentication	
User name	<input type="text" value="DNARMSTRONG"/>
Password	<input type="password" value="....."/>
Confirm password	<input type="password" value="....."/>

Figure 12-8 Enter basic authentication settings for the callback handler

11. Back in the General client policy set bindings → expjeeClient → WS-Security → Authentication and Protection → gen\_signunametoken panel,

panel, click **Save** (in the message panel on the top of the page) to commit the changes.

12. Export the client policy set binding:

- a. In the Admin Console, on the left select Services → Policy sets → General client policy set binding.
- b. On the right, in the General client policy set binding panel, enable the checkbox next to **expjeeClient** and click **Export...** button
- c. In the Application policy sets → Export policy set archive file panel, click the **expjeeClient.zip** link and save in your local file system.

**Behind the scene:**

The exported file contains in its folder structure the bindings.xml file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<securityBindings
  xmlns="http://www.ibm.com/xmlns/prod/websphere/200710/ws-securitybinding">
  <securityBinding name="application">
    <securityOutboundBindingConfig>
      <tokenGenerator
        classname="com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenGenerator"
        name="gen_signunametoken">
        <valueType uri=""
          localName="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken"/>
        <jAASConfig configName="system.wss.generate.unt"/>
        <callbackHandler
          classname="com.ibm.websphere.wssecurity.callbackhandler.UNTGenerateCallbackHandler">
          <basicAuth password="{xor}JycnJycnJyc="
            userid="dnarmstrong"/>
          <properties value="true"
            name="com.ibm.wsspi.wssecurity.token.username.addNonce"/>
          <properties value="true"
            name="com.ibm.wsspi.wssecurity.token.username.addTimestamp"/>
        </callbackHandler>
      </tokenGenerator>
    </securityOutboundBindingConfig>
    <explicitlyProtectSignatureConfirmation/>
  </securityBinding>
</securityBindings>
```

It carries the authentication method (UsernameToken) along with the username and hashed password.



### 12.5.3 Create the provider policy set bindings

The provider policy set bindings instructs the Web service provider on where to find the authentication information.

1. In the Admin Console, on the left select **Services** → **Policy sets** → **General provider policy set bindings**.
2. On the right in the General provider policy set bindings panel, enable the check box next to **Provider Sample** and click **Copy**.
3. On the right in the General provider policy set bindings → Copy of Provider sample panel, set Name to **expjeeProvider** and click **OK**.
4. Back in the Provider policy set bindings list click on the new **expjeeProvider** entry.
5. In the General provider policy set bindings → expjeeProvider panel click the **WS-Security** entry.

#### Behind the scenes:

In the client policy set generation we have removed the other security resources, however here, in the service provider, we do not delete them. This is because that even they are available they are used based on what the client policy set references. So if the client only references the WS-Security resource the other security resources will not be used.

6. In the General provider policy set bindings → expjeeProvider → WS-Security panel, under the Main Message Security Policy Bindings section click **Caller**.
7. In the General provider policy set bindings → expjeeProvider → WS-Security → Callers panel, click **New**.
8. In the General provider policy set bindings → expjeeProvider → WS-Security → Callers → New panel:
  - a. Set Name: **wsnCaller**
  - b. Set Caller identity local part:  
**<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken>**
  - c. Click OK.

### Behind the scenes:

The caller specifies the token or message part that is used for authentication. The Caller identity local part specifies the local name of the caller to use for authentication. The url used above defines the username token type.

Additional information for other token types and their respective Caller identity field values can be found at:

- ▶ WebSphere Application Server Information Center: Caller settings:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/uwbs\\_wsspsbca1d.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/uwbs_wsspsbca1d.html)

9. Back in the General provider policy set bindings → expjeeProvider → WS-Security → Callers panel, click **Save** (in the message panel on the top of the page) to commit the changes.
10. Export the provider policy set binding:
  - a. In the Admin Console, on the left select Services → Policy sets → General provider policy set binding.
  - b. On the right, in the General provider policy set binding panel, enable the checkbox next to **expjeeProvider** and click **Export...** button
  - c. In the Application policy sets → Export policy set archive file panel, click the **expjeeProvider.zip** link and save in your local file system.

### Behind the scene:

The exported policy set bindings zip file contains in its folder structure a bindings.xml file. It contains the definitions needed to consume and decrypt signed and unsigned tokens. In our case we are using an unsigned username token and therefore this is the only relevant section:

```
...
<tokenConsumer
  classname="com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenConsumer
  " name="con_unametoken">
  <valueType uri=""
  localName="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-use
  rname-token-profile-1.0#UsernameToken"/>
  <jAASConfig configName="system.wss.consume.unt"/>
  <callbackHandler
  classname="com.ibm.websphere.wssecurity.callbackhandler.UNTConsumeCall
  backHandler">
  <properties value="true"
  name="com.ibm.wsspi.wssecurity.token.username.verifyTimestamp"/>
    <properties value="true"
  name="com.ibm.wsspi.wssecurity.token.username.verifyNonce"/>
    </callbackHandler>
  </tokenConsumer>
...
```

Additional information about this structure and content of this file can be found at:

- WebSphere Application Server Information Center: Web services security custom properties:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rwbs\\_customproperties.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rwbs_customproperties.html)

## 12.5.4 Import the policy set and binding files

The policy set and binding files created and exported in the previous section must be imported into the workbench to be available for use with Web service providers and clients.

1. Import the policy file.
  - a. In the workbench action bar, select **File** → **Import**.

- b. At the Import pop-up, select **Web Services** → **WebSphere Policy Sets** and click **Next**.
    - c. At the Policy Set Import Wizard pop-up use the **Browse...** button to select the **UsernameMinimum.zip** policy set file exported in “12.5.1 Create the policy set” on page 454 and click **Finish**.
  2. Import the policy set bindings files.
    - a. In the workbench action bar, select **File** → **Import**.
    - b. At the Import pop-up, select **Web Services** → **WebSphere Named Bindings** and click **Next**.
    - c. At the Named Binding Import Wizard pop-up, use the **Browse...** button to select the **expjeeProvider.zip** bindings set file exported in “12.5.3 Create the provider policy set bindings” on page 461 and click **Finish**.
    - d. Repeat the steps above for importing the client policy set bindings file (expjeeClient.zip).
  3. Confirm the correct import of security files
    - a. In the workbench action bar, select **Window** → **Preferences**.
    - b. At the Preferences pop-up, on the left select **Service Policies**.
    - c. On the right at the resulting Service Policies pane:
      - i. Expand **WebSphere General Bindings** and the two sub nodes: **WebSphere Client General Bindings** and **WebSphere Server General Bindings**.
      - ii. Expand **WebSphere Policy Sets** → **WebSphere v7.0 Policy Sets**.
      - iii. Verify that all three entries appear.

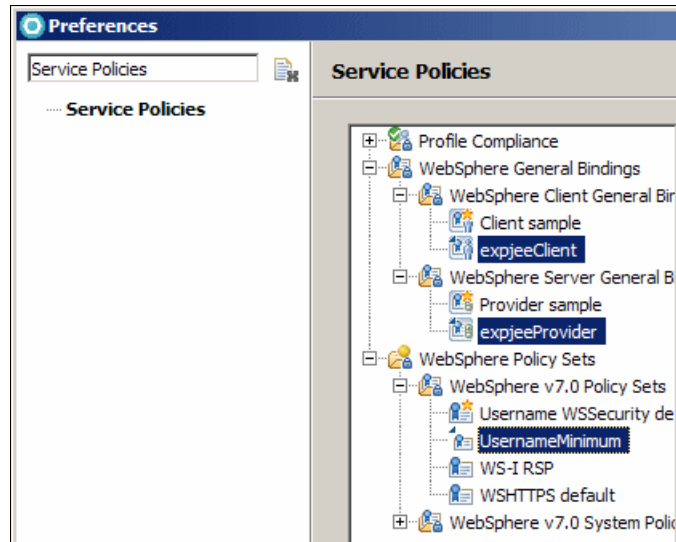


Figure 12-9 Service policies imported to the workspace

## 12.5.5 Attach the policy set to the Web service provider

We now enable the security to be used with the Web service provider in the workbench by attaching the policy set and binding file.

### Behind the scene:

By attaching the policy set configurations in the workbench we are allowing the workbench to deploy a project to the server referencing the WebSphere Application Server configurations.

1. From the services view select the **DonateEJB: {...}** service (under JAX-WS) and right click **Manage Policy Set Attachment**.
2. At the Service Side Policy Set Attachment ... pop-up, click **Add**.
3. At the End Point Definitions Dialog pop-up:
  - a. Accept the default values for Service Name and endpoint.
  - b. Set Policy Set to **UsernameMinimum**
  - c. Set Binding to **expjeeProvider**
  - d. Click **OK**.

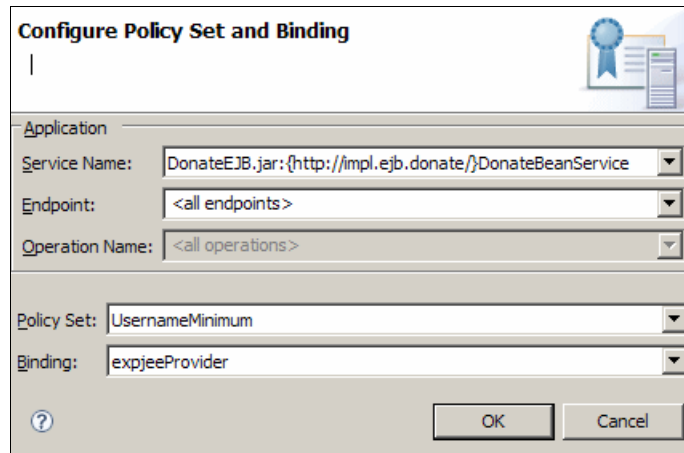


Figure 12-10 Policy set and binding configuration for DonateEJB

4. Back at the Service Side Policy Set Attachment ... pop-up, confirm that the policy was added and click **Finish**.

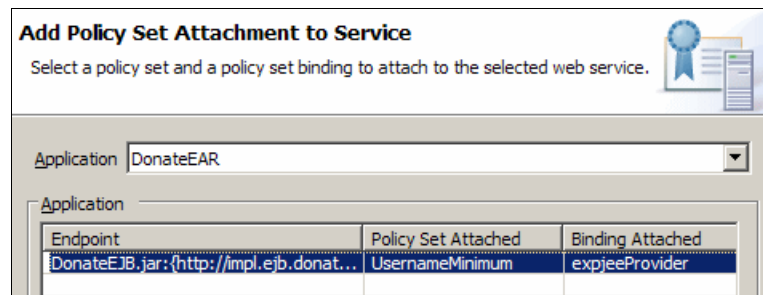


Figure 12-11 New policy set has been added

## 12.5.6 Attach the policy set to the Web service client.

We now enable the security to be used with the Web service client in the workbench by attaching the policy set and binding files to the Web service.

1. From the Services view select **DonateWSClient :{...}** (under **JAX-WS → Clients**) and right click **Manage Policy Set Attachment**.
2. At the Client Side Policy Set Attachment ... / Configure Policy acquisition for Web Service Client, accept the default values and click **Next**.
3. At the Client Side Policy Set Attachment ... / Add Policy Set Attachment to Web Service Client pop-up, click **Add**.
4. In the End Point Definition Dialog pop-up:

- a. Accept the default value for Service Name.
- b. Set Endpoint: **DonateBeanPort**
- c. Set Operation Name: **donateToFund**
- d. Set Policy Set: **UsernameMinimum**
- e. Binding: **expjeeClient**
- f. Click **OK**.

Figure 12-12 Policy set and binding configuration for DonateWSCClient

5. Back at the Client Side Policy Set Attachment ... pop-up, click **Finish**.

#### Off course:

If you have already attached any policy set and have not enable automatic file overwriting in the workbench, you will be prompted to overwrite the clientPolicyAttachments.xml file of the client project. In this dialog you can enable it to enable automatic file overwriting.

If you repeat these sections you may be prompted to overwrite the policy binding files for wsPolicyClientControl.xml and clientPolicyAttachments.xml (under DonateWSCClientEAR/META-INF) for the client) and policyAttachments.xml (under DonateEAR/META-INF) for the service.

## 12.5.7 Test the secure Web service using the DonateWSCClient

In this section we use the Web services client created in a servlet to test the secured access. The TCP/IP monitor is also used in order to capture the messages exchanged by the client and the server. If your WS client was not configured with the URL to use the TCP/IP monitor, review 11.7.1, “Monitor the SOAP traffic using TCP/IP Monitor” on page 437).

1. Publish the DonateWSCClientEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. In the Java EE perspective, select **TestClient.jsp** (in DonateWSCClient/WebContent/sampleDonateBeanPortProxy) and right click **Run As** → **Run on Server**.
3. In the resulting Web Service Test Client page, under the Methods pane (on the left) select **donateToFund**.
4. In the resulting donateToFund pane (on the right):
  - arg0: **1**
  - arg1: **DonationFund**
  - arg2: **1**
  - a. Click **Invoke**.
5. In the Result pane, note the successful invocation Transfer successful



### Extra Credit:

The TCP/IP Monitor can be used to display the entire SOAP request, including the WS-security header:

```
<soapenv:Header>
  <s:Security soapenv:mustUnderstand="1"
    xmlns:s="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssec
    urity-secext-1.0.xsd"
    xmlns:u="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssec
    urity-utility-1.0.xsd">
    <s:UsernameToken u:Id="unt_20">
      <s:Username>dnarmstrong</s:Username>
      <s:Password
        Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username
        -token-profile-1.0#PasswordText">new42day</s:Password>
        <u:Created>2010-01-25T23:21:11.234Z</u:Created>

      <s:Nonce>uD5HA0qsufIEWNdIXoHy1Tq9wyXSdVbBA1Zx72JxwTd0cZnNI+U4zZd3L+nd9
      TLPHTxnkgG/rXsob00d4Nmrc0qC0cIzVKMVh3uVprbHHVDu5/RmZpGHJgDbf9t1HLIjwF
      RbvcYhV9Ya5DPZHb7XE46jCB1EJpBDfiX1Bz9QiA=</s:Nonce>
    </s:UsernameToken>
  </s:Security>
</soapenv:Header>
<soapenv:Body>
  <ns2:donateToFund xmlns:ns2="http://impl.ejb.donate/">
    <arg0>1</arg0>
    <arg1>DonationFund</arg1>
    <arg2>1</arg2>
  </ns2:donateToFund>
</soapenv:Body>
</soapenv:Envelope>
```

### Extra credit:

In the section above we have used DNARMSTRONG userid to test the secured service.

We have two different user IDs to demonstrate role based security:

- ▶ DNARMSTRONG is granted access and the first Web service request returns Transfer successful.
- ▶ DGADAMS is denied access and the Web service request should return an Unauthorized error.

To test it create a client policy set bindings with the DGADAMS credentials using the section 12.5.2, “Create the client policy set bindings” on page 458 as a guidance.

Then import the policy set bindings file using the section 12.5.6, “Attach the policy set to the Web service client.” on page 466.

Publish and restart the server. Run the DonateWSCClient TestClient.jsp again.

The result is:

```
EXCEPTION: java.rmi.AccessException: ; nested exception is:
com.ibm.websphere.csi.CSIAccessException: SECJ0053E: Authorization failed
for defaultWIMFileBasedRealm/DGADAMS while invoking
(Been)DonateEAR#DonateEJB.jar#DonateBean
donateToFund:int,java.lang.String,int:5 Subject: Principal:
defaultWIMFileBasedRealm/DGADAMS Public Credential:
com.ibm.ws.security.auth.WSCredentialImpl@2ab32ab3 Private Credential:
com.ibm.ws.wssecurity.platform.websphere.wssapi.token.impl.WasUsernameTok
enImpl:DGADAMS Private Credential:
com.ibm.ws.security.token.SingleSignonTokenImpl@307f307f Private
Credential: com.ibm.ws.security.token.AuthenticationTokenImpl@5e6a5e6a
Private Credential:
com.ibm.ws.security.token.AuthorizationTokenImpl@30253025 is not granted
any of the required roles: DMANAGERS_ROLE DUSERS_ROLE
```

## 12.6 Explore!

In this section we describe a few Explore activities.

## 12.6.1 Additional resources

Additional Learn resources are available at the following Web sites:

- ▶ WS-Notification in WebSphere Application Server V7: Part 2: Configuring JAX-WS applications with WS-Security for WS-Notification  
[http://www.ibm.com/developerworks/websphere/techjournal/0904\\_jiang/0904\\_jiang.html](http://www.ibm.com/developerworks/websphere/techjournal/0904_jiang/0904_jiang.html)
- ▶ Message-level security with JAX-WS on WebSphere Application Server v7  
[http://www.ibm.com/developerworks/websphere/tutorials/0905\\_griffith/index.html](http://www.ibm.com/developerworks/websphere/tutorials/0905_griffith/index.html)
- ▶ Web Services Security UsernameToken Profile 1.0:  
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>
- ▶ See “Chapter 8. Web services security”, in *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257, located at:  
<http://www.redbooks.ibm.com/abstracts/sg247257.html>

## 12.6.2 Secure Web services using transport level security

The WS-Security implementation described in this chapter inserts the caller identity into the actual SOAP message. The caller identity can also be passed using the HTTP BASIC authentication mechanism used to protect Web container resources.

Web services best practices typically discourage using HTTP BASIC authentication because it supports a limited number of identity representations, it is based at the transport level (which can cause complications if the SOAP request transits through an intermediary such as an Enterprise Service Bus), and it is an HTTP transport specific capability (and Web services can be supported over other transports such as JMS).

However, this approach may be sufficient for certain environments and is popular in the context of Web 2.0 (which will be shown in Chapter 16, “Develop a Web 2.0 client” on page 547).

This security implementation is done in the Redbook: *Experience Java EE! Using WebSphere Application Server Community Edition 2.1*, Chapter 12: Implement security for the Web service:

<http://www.redbooks.ibm.com/abstracts/SG247639.html>

## 12.6.3 Secure Web services with DataPower

What is DataPower®?

From `ibm.com`:

IBM WebSphere DataPower SOA Appliances represent an important element in IBM's holistic approach to Service Oriented Architecture (SOA). IBM SOA appliances are purpose-built, easy-to-deploy network devices that simplify, help secure, and accelerate XML and Web services deployments, while extending your SOA infrastructure. These new appliances offer an innovative, pragmatic approach to harness the power of SOA while simultaneously enabling you to leverage the value of your existing application, security, and networking infrastructure investments.

There are actually three different DataPower appliances offered by IBM:

- ▶ **Integration Appliance X150** is used for integration into back-end systems, including legacy.
- ▶ **XML Accelerator XA35** is used to off-load XML, XSD, XPath, and XSLT processes from over-tasked servers to this highly efficient appliance.
- ▶ **XML Security Gateway XS40** was built by some of the world's top Web services experts to deliver secured Web services transactions.

The remainder of this section is dedicated to the XML Security Gateway XS40 which provides a security-enforcement point for XML and Web services transactions, including encryption, firewall filtering, digital signatures, schema validation, WS-Security, XML access control, XPath and detailed logging, and includes:

- ▶ **XML/SOAP firewall:** The DataPower XML Security Gateway XS40 filters traffic at wire speed, based on information from layers 2 through 7 of the protocol stack; from field-level message content and SOAP envelopes to IP address, port/hostname, payload size, or other metadata. Filters can be predefined with easy point-and-click XPath filtering GUI, and automatically uploaded to change security policies based on the time of day or other triggers.
- ▶ **XML/SOAP data validation:** With its unique ability to perform XML Schema validation as well as message validation, at wire speed, the XS40 ensures incoming and outgoing XML documents are legitimate and properly structured. This protects against threats such as XML Denial of Service (XDoS) attacks, buffer overflows, or vulnerabilities created by deliberately or inadvertently malformed XML documents.

- ▶ **Field level message security:** The XS40 selectively shares information through encryption/decryption and signing/verification of entire messages or of individual XML fields. These granular and conditional security policies can be based on nearly any variable, including content, IP address, hostname or other user-defined filters.
- ▶ **XML Web services access control:** The XS40 supports a variety of access control mechanisms, including WS-Security, WS-Trust, X.509, SAML, SSL, LDAP, RADIUS and simple client/URL maps. The XS40 can control access rights by rejecting unsigned messages and verifying signatures within SAML assertions.
- ▶ **Service virtualization:** XML Web services require companies to link partners to resources without leaking information about their location or configuration. With the combined power of URL rewriting, high-performance XSL transforms, and XML/SOAP routing, the XS40 can transparently map a rich set of services to protected back-end resources with high performance.
- ▶ **Centralized policy management:** The XS40's wire speed performance enables enterprises to centralize security functions in a single drop-in device that can enhance security and help reduce ongoing maintenance costs. Simple firewall functionality can be configured via a GUI and running in minutes, and using the power of XSLT, the XS40 can also create sophisticated security and routing rules. Because the XS40 works with leading Policy Managers, it is an ideal policy execution engine for securing next generation applications. Manageable locally or remotely, the XS40 supports SNMP, script-based configuration, and remote logging to integrate seamlessly with leading management software.
- ▶ **Web services management/service level management:** With support for Web services Distributed Management (WSDM), Universal Description, Discovery, and Integration (UDDI), Web Services Description Language (WSDL), Dynamic Discovery, and broad support for Service Level Management configurations, the XS40 natively offers a robust Web services management framework. This framework allows the efficient management of distributed Web service endpoints and proxies in heterogeneous SOA environments, as well as SLM alerts and logging, and pull and enforce policies, which helps enable broad integration support for third-party management systems and unified dashboards, in addition to robust support and enforcement for governance frameworks and policies.

For more information, refer to the DataPower Web site:

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





## Part 4

# Messaging

In Part 4, we extend the core Java EE application to support JMS messaging, by performing the tasks described in the following chapters:

- ▶ Chapter 13, “Create the message-driven bean” on page 477: Create a message-driven bean to receive messages and to call the Donate session EJB. Create a JMS client to send a message to the message-driven bean.
- ▶ Chapter 14, “Add publication of results” on page 517: Update the Donate session EJB to publish the donation request results to a topic representing the employee number, and create a JMS client (to subscribe to updates for an employee).
- ▶ Chapter 15, “Implement security for messaging” on page 535: Update the messaging artifacts to support security.







## Create the message-driven bean

In this chapter you perform the following actions:

- ▶ Create a message-driven bean to receive messages and call the Donate session EJB.
- ▶ Create a JMS client to send a message to the message-driven bean.

## 13.1 Learn!

Figure 13-1 shows the Donate Application with its message driven bean and JMS client.

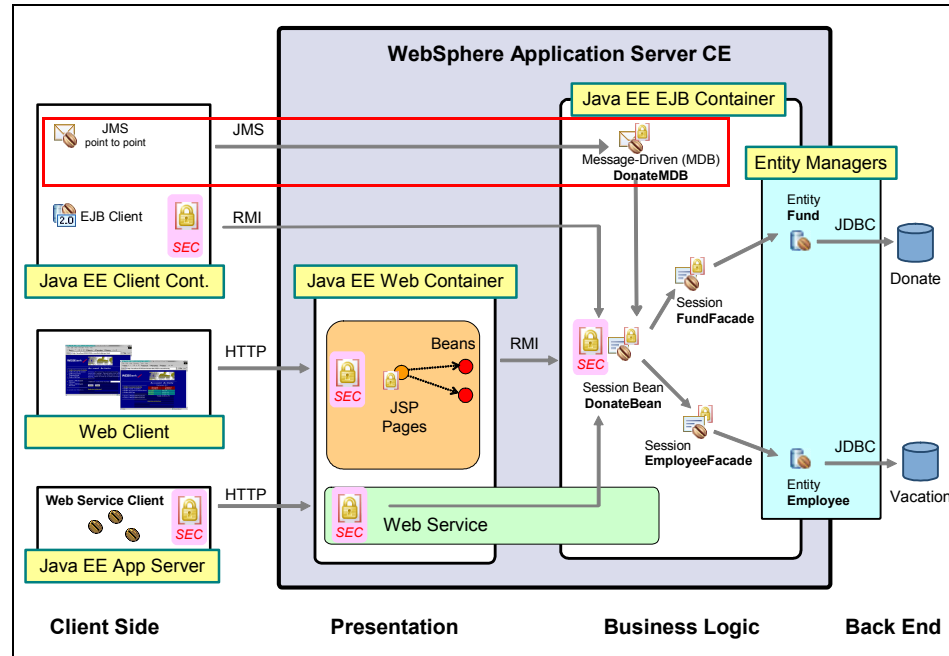


Figure 13-1 Donate Application: Message-driven bean and JMS client

Messaging provides an asynchronous communication mechanism that is an alternative to HTTP for program-to-program communication:

- ▶ HTTP is synchronous, similar in concept to an instant message application. The sender and receiver both must be active.

This makes for a tightly coupled solution. Applications are aware of and must handle communication errors, whether short term (milliseconds) or longer term (seconds and higher). As a result, an outage in any one part—whether due to networking errors, hardware failures, application failures, or maintenance and upgrades—can cause delays across the broader system due to the cascading delays of error handling, waits, and retries.

- ▶ Messaging is asynchronous, similar in concept to an e-mail application. The sender creates the message, and gives it to a messaging subsystem, which accepts it for delivery.

The messaging subsystem takes care of sending the message to the target system—handling any networking errors or system shutdowns—and the application on the target system receives the message, either auto-started by the messaging system or when the application makes a call. The application can optionally reply, sending a similar asynchronous message back to the requester.

This allows for a loosely coupled solution: Applications only have to be aware that the message was “accepted” for delivery and then can proceed to other processing. An outage in any one part does not immediately block operation in the rest of the system.

Java EE contains the Java Message Service (JMS) specification, which defines an API that applications can utilize to access a messaging provider (a product implementation that provides the infrastructure that supports messaging, such as Active MQ or IBM WebSphere MQ).

JMS is the API specification only and does not describe the actual messaging product or provider. This is similar to accessing relational databases through JDBC, where JDBC defines the access mechanism but does not define the actual relational database product.

All the interaction with the Java EE server so far was over HTTP. How do you access Java EE over messaging?

- **Client side:** A program creates a message and sends it to a JMS queue, in this case, a local queue (Figure 13-2). The message stays on the source system and is not forwarded to a remote system, and then terminates. The program does not know when (or if) the message is actually read, only that the message is persisted in the queue.

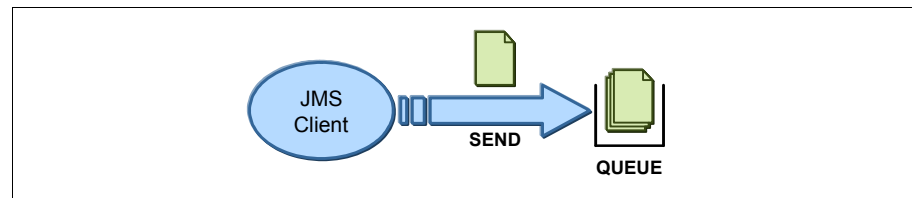


Figure 13-2 Accessing Java EE over messaging - client side

In this scenario, the program is a standalone Java program, but you could put the same basic code inside Java EE server side artifacts (such as servlets or EJBs) if you wanted to send a message from one Java EE server to another, or even to another component in the same Java EE server.

- **Java EE server side:** An EJB container is configured to listen to a queue through a definition called an activation specification. An EJB artifact called a message-driven bean (MDB) is created, with a method called `onMessage` that contains the user defined business logic, and is deployed to this EJB container, referencing the same activation specification (Figure 13-3). When the EJB container starts, it opens the activation specification and issues a receive against the queue. When a message is placed on the queue, the activation specification receives the message and calls the associated MDB `onMessage` method, which runs the user defined business logic.

Note that MDBs have little in common with entity EJBs and session EJBs except for executing in the same Java EE container. The MDB is invoked by the overall Java EE infrastructure and does not have any external local or remote interfaces.

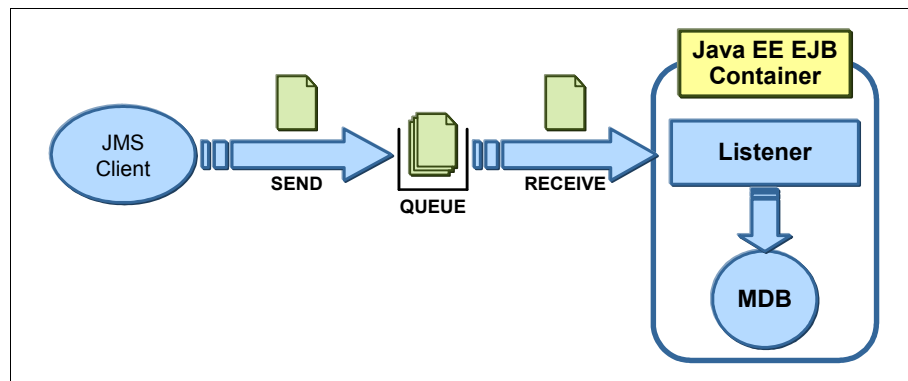


Figure 13-3 Accessing Java EE over messaging - server side

### 13.1.1 Additional resources

The following additional Learn resources are available:

- J2EE: Java Message Service API – FAQ:  
<http://java.sun.com/products/jms/faq.html>
- IBM developerWorks: J2EE pathfinder: Enterprise messaging with JMS:  
<http://www.ibm.com/developerworks/java/library/j-pj2ee5/>
- Redbook: WebSphere Application Server V7 Messaging Administration Guide:  
<http://www.redbooks.ibm.com/redbooks.nsf/RedpieceAbstracts/sg247770.html>
- WebSphere Application Server V7.0 InfoCenter:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6tech\\_si.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/welc6tech_si.html)

### 13.1.2 EJB 3.0 message-driven beans

Message-driven beans are used for the processing of asynchronous JMS messages within Java EE based applications. They are invoked by the container on the arrival of a message.

In this way, message-driven beans can be thought of as another interaction mechanism for invoking EJBs, but unlike session beans, the container is responsible for invoking them when a message is received, not a client (or another bean).

To define a message driven bean in EJB 3.0 we declare a POJO with the `@MessageDriven` annotation:

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName="destinationType",
                               propertyValue="javax.jms.Queue"),
    @ActivationConfigProperty(propertyName="destination",
                               propertyValue="queue/myQueue")
})
public class MyMessageBean implements javax.jms.MessageListener {

    public void onMessage(javax.msg.Message inMsg) {
        // implement the onMessage method
        // to handle the incoming message
        ....
    }
}
```

The main relevant features of this example are:

- ▶ In EJB 3.0, the MDB bean class is annotated with the **@MessageDriven** annotation, which specifies a set of activation configuration parameters. These parameters are unique to the particular kind of JCA 1.5 adapter that is used to drive the MDB. Some adapters have configuration parameters that let you specify the destination queue of the MDB. In the case where the adapter does not support this, the destination name must be specified using a `<message-destination>` entry in the XML binding file.
- ▶ The bean class has to implement the **MessageListener** interface, which defines only one method, **onMessage**. When a message arrives in the queue monitored by this MDB, the container calls the `onMessage` method of the bean class and passes the incoming message in as the parameter.
- ▶ Furthermore, the `activationConfig` property of the `@MessageDriven` annotation provides messaging system-specific configuration information.

### 13.1.3 Service integration bus

WebSphere Application Server V6 introduced a service integration bus (SIB) that provides an infrastructure for supporting the exchange of messages with other systems:

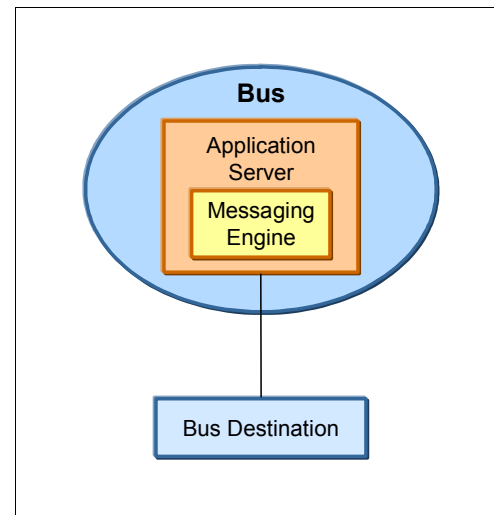
- Messages can be sent and received over multiple protocols—base JMS, Web services over HTTP, and Web services over JMS. Messages can be received in one protocol, and resent in another.

Messages can be mediated, meaning that they can be routed to alternate destinations or the content updated and transformed.

This type of capability was available for many years, but usually in separate implementations for each protocol—one product with transformation and routing for messaging (whether JMS or native messaging) and another product with transformation and routing for Web services. This combined capability is generically called an Enterprise Service Bus (ESB).

You are using a limited set of functions with the SIB, implementing a single server bus to support basic JMS operations:

- **Bus destination:** Represents an input or output location (JMS queue, JMS topic, Web service).
- **Messaging engine:** Manages the interaction between user applications and message destinations, providing message persistence, transaction context, transformation, and routing.
- **Service integration bus:** Provides both a configuration and runtime grouping of one or more Application Servers. A managed WebSphere environment may contain one or more buses allowing for separation of messaging by organization or function.



An Application Server contains one messaging engine for each bus to which it belongs and can directly exchange JMS messages to other Application Servers on the same bus.

## 13.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.


## 13.3 Disable Java EE EJB declarative security

This chapter and the subsequent chapter (Chapter 14, “Add publication of results” on page 517) do not contain the application code required to support a secure session EJB. Therefore, before starting this chapter, we have to disable the Java EE EJB declarative security. We will re-enable EJB security at the start of Chapter 15, “Implement security for messaging” on page 535.

1. Disable declarative security in the `DonateBean` session EJB:
  - a. In the Enterprise Explorer, open **DonateBean** (in `DonateEJB/ejbModule/donate.ejb.impl`).
  - b. Before the two `donateToFund` methods, comment out the `@RolesAllowed` statement and activate the `@PermitAll` statement.

```
//@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
@PermitAll
public String donateToFund(Employee employee, .....)
```

```
//@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
@PermitAll
@WebMethod
public String donateToFund(int employeeId, .....)
```

- c. Organize imports (**Ctrl+Shift+O**) to resolve `javax.annotation.security.PermitAll`.
    - d. Save and close `DonateBean.java`.
  2. Restart the server to ensure that all active instances of the application are running with security disable.
    - If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
    - If the **ExperienceJEE Test Server** is running, select the server in the Servers view, right-click and select **Restart**.

- If the server instance starts (restarts) successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.





## 13.4 Configure the test server for base JMS resources

In this section you configure the ExperienceJEE Test Server with the required resources for the MDB:


- ▶ Create the base SIB definition in 13.4.1, “Create the Service Integration Bus” on page 484.
- ▶ Configure security in the SIB in 13.4.3, “Create the destination queue resource” on page 493
- ▶ Create the Donate queue in the SIB in 13.4.3, “Create the destination queue resource” on page 493.
- ▶ Create the JMS queue definition that provides the link to the Donate queue in the SIB in 13.4.4, “Create the JMS queue destination” on page 496.
- ▶ Create the JMS activation specification (needed for the MDB) in 13.4.5, “Create the JMS activation specification” on page 498.
- ▶ Create the JMS queue connection factory in 13.4.6, “Create the JMS connection factory for the application client” on page 500.

This resource is not used by the MDB but instead is used by the Java EE application client defined later in this chapter and the pub sub logic added in the subsequent chapter. It is defined here to keep all JMS related definitions in one section.

### 13.4.1 Create the Service Integration Bus

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to:
    -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`
  - and execute:
    -  `startNetworkServer.bat`
    -  `startNetworkServers.sh`



- If the ExperienceJEE Test Server is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
  - If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.

2. Start the Admin Console using the same steps as describe in 4.6.1, “Start the Administrative Console” on page 100.

Recall that in the Admin Console login screen you used the javaeeadmin user ID and password that you defined in 3.4, “Create the ExperienceJEE server profile” on page 62.

3. In the Admin Console, on the left select **Service integration** → **Buses**.
4. On the right, in the **Buses** panel, click **New**.

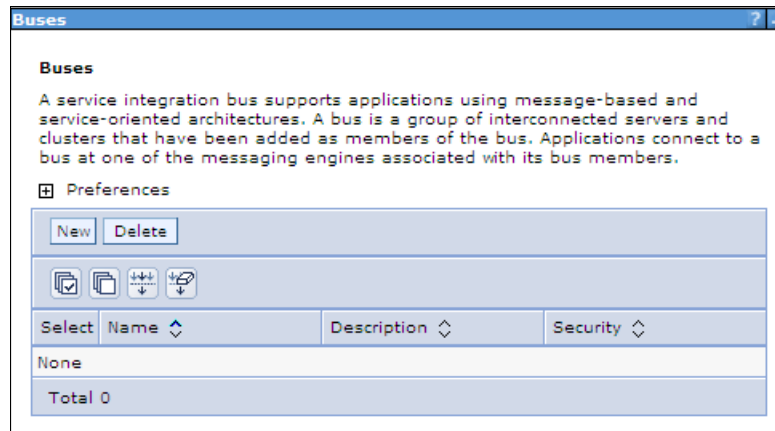


Figure 13-4 Create a new bus

- a. On the right, in the Create a new messaging engine bus panel:
  - i. Enter the name for your new bus: **ExperienceJEE**
  - ii. Leave Bus security enabled.
- b. Click **Next**.

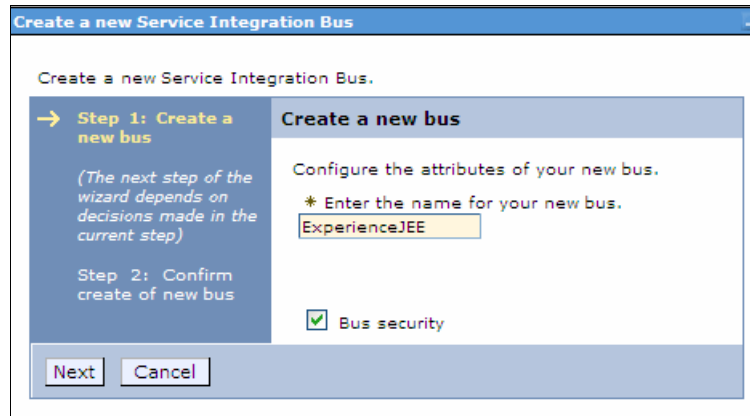


Figure 13-5 Create a new bus - step 1

- c. On the right, in the Configure Bus Security Panel / Step 1.1: Introduction panel, click **Next**.
- d. On the right, in the Configure Bus Security Panel / Step 1.2: Specify transport level security panel:
  - i. Clear the Require clients use SSL protected transports selection
  - ii. Click **Next**.

**Behind the scenes:** This option is appropriate for production environments where it is critical to protect data confidentially. However, it does add an additional complexity to configuring a basic development and test environment that information that is beyond the scope of this book. Refer to the Redbook: WebSphere Application Server V7 Messaging Administration Guide for further details on this topic:

<http://www.redbooks.ibm.com/redbooks.nsf/RedpieceAbstracts/sg247770.html>

- e. On the right, in the Configure Bus Security Panel / Step 1.3: Select the security domain for the Bus panel:
  - i. Enable the **Use global security domain** selection.
  - ii. Click **Next**.

<p>Step 1: Create a new bus</p> <p>Step 1.1: Introduction</p> <p>Step 1.2: Specify transport level security</p> <p>→ <b>Step 1.3: Select the security domain for the bus</b></p> <p>(The next step of the wizard depends on decisions made in the current step)</p> <p>Step 1.4: Confirm the enablement of security</p>	<p><b>Select the security domain for the bus</b></p> <p>You are configuring a bus that consists of bus members that are all Version 7. You can therefore configure the bus to use a security domain other than the global security domain.</p> <p>If the bus uses a domain other than the global security domain then you cannot add pre Version 7 members to this bus.</p> <p><input type="radio"/> Use the global security domain</p> <p><input checked="" type="radio"/> Inherit the cell level security domain</p> <p><input type="radio"/> Use an existing security domain (none) ▼</p> <p><input type="radio"/> Create a new security domain</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 13-6 Create a new bus - step 1.3

#### Behind the scenes:

The ExperienceJEE Test Server has one security domain that is used for both application security and application server administration (and bus) security. This is the global security option.

WebSphere Application Server v7.0 allows you to configure separate security domains for application security, application server administration, and bus security. In those cases, the other options would be appropriate (Inherit the cell level security domain, Use an existing security domain, or Create a new security domain).

From a practical standpoint, since your ExperienceJEE Test Server has a single domain, both the default option (Inherit the cell level security domain) and the selected option (Use the global security domain) have the same effect.

- f. On the right, in the Configure Bus Security Panel / Step 1.4: Confirm the enablement of security panel, click **Next**.
- g. On the right, in the Configure Bus Security Panel / Step2: Confirm create of the new bus panel, click **Finish**.
5. Back In the Buses panel, which is updated with the new entry, scroll down, and click the **ExperienceJEE** bus.

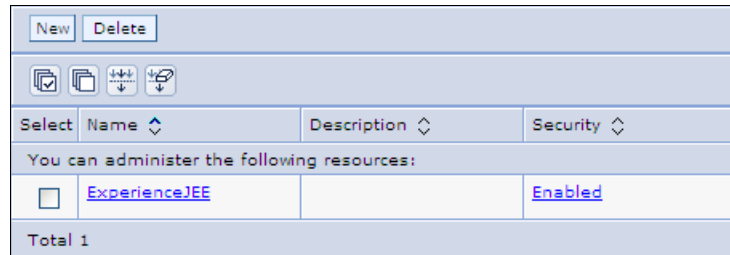


Figure 13-7 New bus

- On the right, in the Buses → ExperienceJEE panel, under Topology, click **Bus members**.

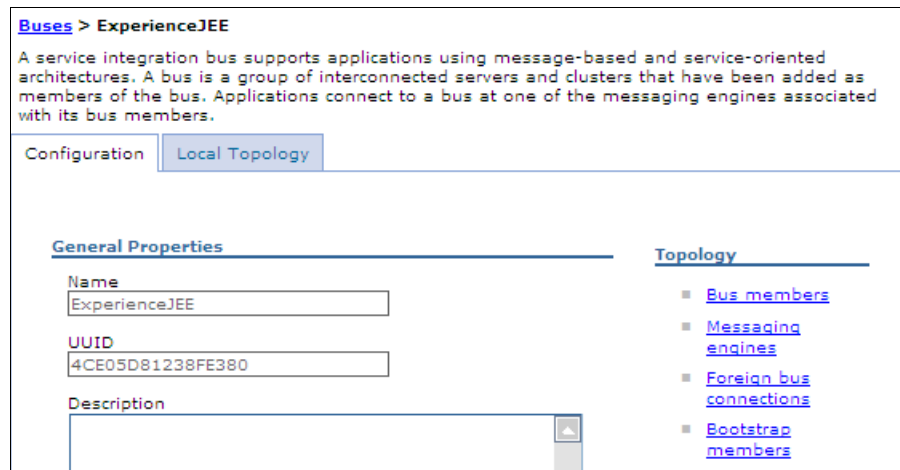


Figure 13-8 Select Bus members

- On the right, in the Buses → ExperienceJEE → Bus members panel, click **Add**.



Figure 13-9 Add a new bus member

- a. In the **Step 1**: Select server, cluster, or WebSphere MQ Server panel:
  - i. Ensure that the **Server** option is enabled and that the server is **ExperienceJEE:server1**,
  - ii. Click **Next**.

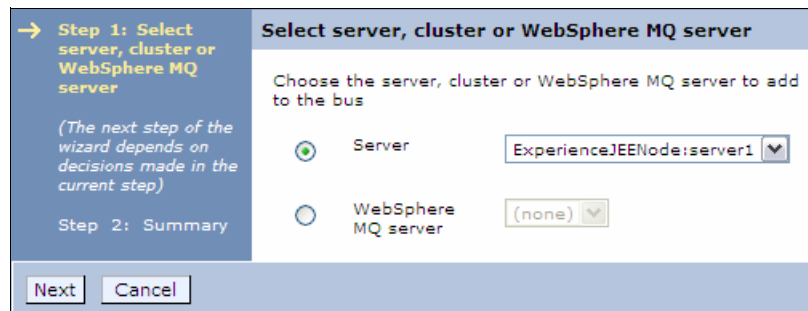


Figure 13-10 Select the server as the new member

- b. In the Step 1.1: Select the type of message store panel, accept the default selection of **File store**, and click **Next**.
- c. In the Step 1.2: Configure file store panel, accept the default selections and click **Next**.
- d. In the Step 1.3: Tune performance Parameters panel click **Next**.
- e. In the Step 2: Summary panel, click **Finish**.

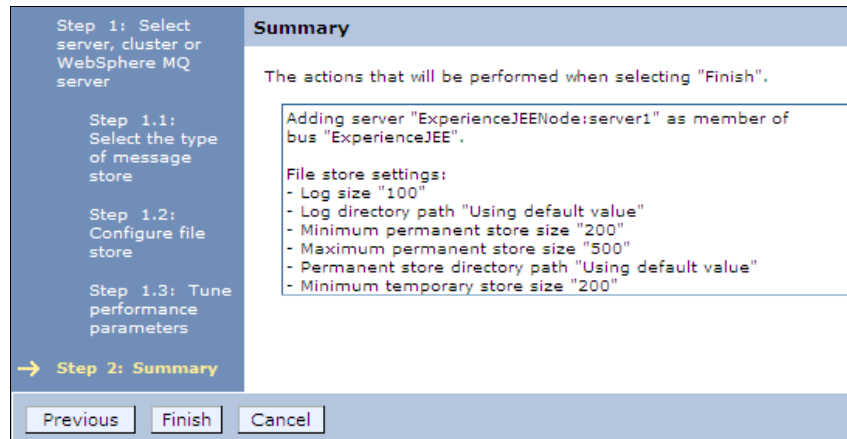


Figure 13-11 New bus member settings

- At the Buses → ExperienceJEE → Bus Members panel, verify that bus member was added:

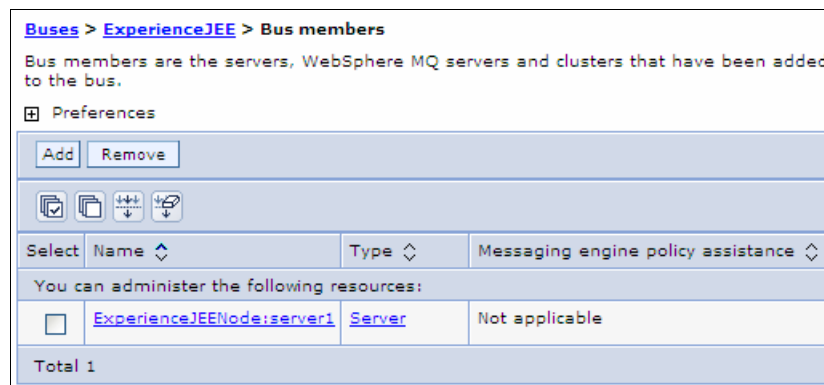


Figure 13-12 New bus member

## 13.4.2 Configure security

Since the bus was enabled with security enabled, a user ID and password that maps to an access role (similar to that used for Java EE application security) is required to connect to the bus in order to send and receive messages.

Therefore, you will configure the bus to allow access by either the javaeeadmin user or by users that belong to the DUSERS group (defined in 10.3, "Preliminary security setup" on page 365).

- In the Admin Console, on the left select **Service integration** → **Buses**.

2. On the right, at the Buses panel, click **ExperienceJEE**.
3. On the right, at the Buses → ExperienceJEE panel, under Additional properties click **Security**. Note that you may need to scroll down to see this selection.
4. On the right at the Buses → ExperienceJEE → Security for bus ExperienceJEE panel, under Authorization Policy click **Users and groups in the bus connector role**.

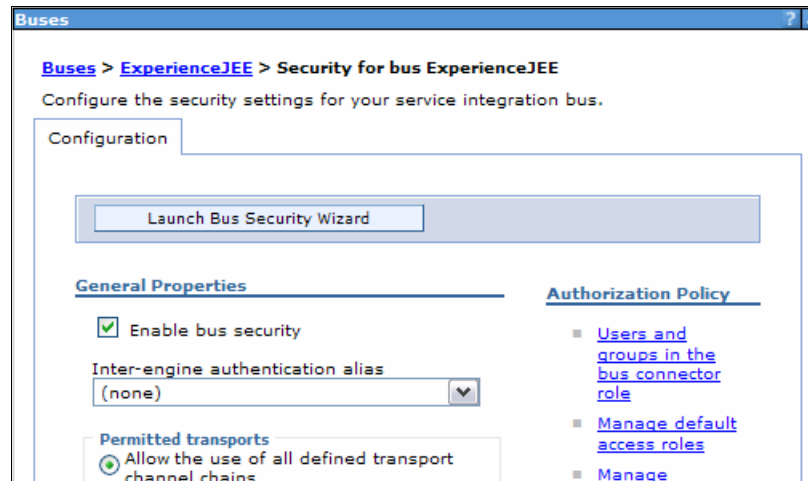


Figure 13-13 Add a users in the bus connector role - step 1

5. On the right, at the Buses → ExperienceJEE → Security for bus ExperienceJEE → Users and groups in the bus connector role panel, click **New**.

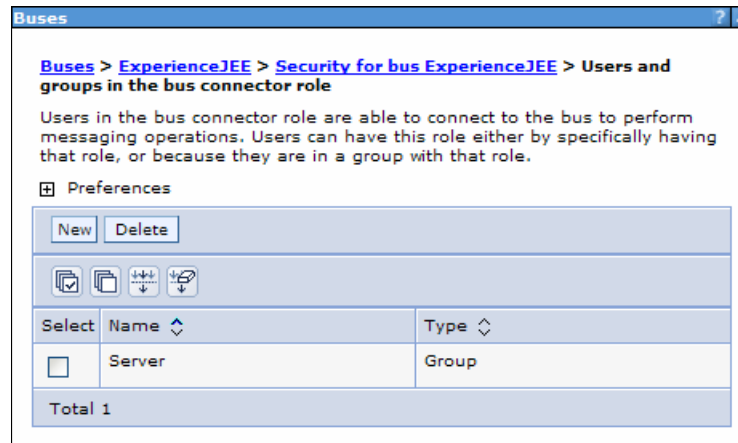


Figure 13-14 Add a users in the bus connector role - step 2

- a. On the right, at the SIB Security Resource Wizard / Step 1 panel, select **Users** and click **Next**.
- b. On the right at the SIB Security Resource Wizard Step 2 panel, select **javaeeadmin** and click **Next**.
- c. On the right at the SIB Security Resource Wizard Step 3 panel, click **Finish**.
6. On the right, at the Buses → ExperienceJEE → Security for bus ExperienceJEE → Users and groups in the bus connector role panel, click **New**.
  - a. On the right, at the SIB Security Resource Wizard / Step 1 panel, select **Groups** and click **Next**.
  - b. On the right at the SIB Security Resource Wizard Step 2 panel, select **DUSERS** and click **Next**.
  - c. On the right at the SIB Security Resource Wizard Step 3 panel, click **Finish**.
7. Back at the Buses → ExperienceJEE → Security for bus ExperienceJEE → Users and groups in the bus connector role panel, verify that the roles were added.







New Delete		
   		
Select	Name	Type
<input type="checkbox"/>	DUSERS	Group
<input type="checkbox"/>	Server	Group
<input type="checkbox"/>	javaeeadmin	User
Total 3		

Figure 13-15 New users and groups in the bus connector role

- At the top of the panel, in the Messages box that states “Changes have been made to your local configuration...”, click **Save**.

### 13.4.3 Create the destination queue resource

A destination in the SIB provides the physical resource that holds messages. Destinations are accessed in a persistent mode (meaning that messages are preserved across reboots) or a non-persistent mode (meaning that messages are not preserved across reboots).

- In the Admin Console, on the left select **Service integration** → **Buses**.
- On the right, in the Buses panel, click **ExperienceJEE**.

New Delete			
   			
Select	Name	Description	Security
You can administer the following resources:			
<input type="checkbox"/>	ExperienceJEE		Enabled
Total 1			

Figure 13-16 Select the bus

- On the right in the Buses → ExperienceJEE panel, under Destination resources, click **Destinations**.

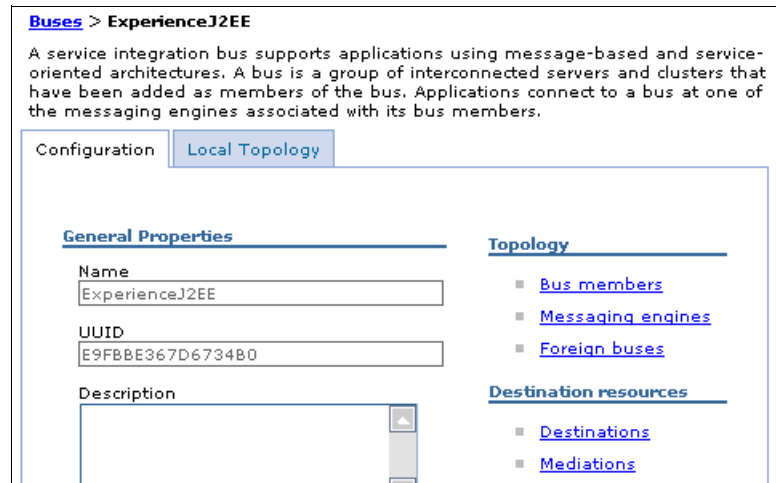


Figure 13-17 Open the destinations

4. On the right in the Buses → ExperienceJEE → Destinations panel, click **New**.



Figure 13-18 Add a new destination

- a. In the Create a new destination panel, ensure that Select destination type is set to **Queue**, and click **Next**.

**Create new destination**

Create a new destination on this bus.

**Select destination type**

- ☒ Queue
- ☐ Topic space
- ☐ Alias
- ☐ Foreign

Figure 13-19 Select queue as the destination type

- b. In the Step1: Set queue attributes panel, set Identifier to **DonateQ**, and click **Next**.

**Create new queue**

Create a new queue for point-to-point messaging.

→ **Step 1: Set queue attributes**

Step 2: Assign the queue to a bus member

Step 3: Confirm queue creation

**Set queue attributes**

Configure the attributes of your new queue

\* Identifier:

Description:

Figure 13-20 Set the queue name

- c. In the Step 2: Assign the queue to a bus member panel, verify that the bus is set to **Node=ExperienceJEENode:Server1=server1**, and click **Next**.
- d. In the Create new queue (Step 3: Confirm queue creation) panel, click **Finish**.
5. Back in the Buses → ExperienceJEE → Destinations panel, verify that the queue destination was added.



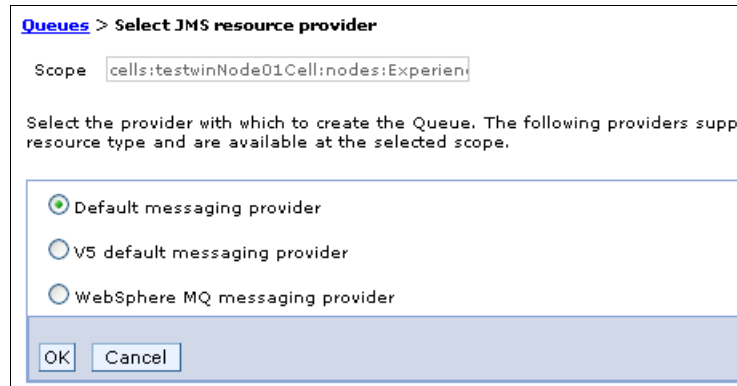


Figure 13-23 Select the messaging provider

5. On the right, in the Queues → Default messaging provider → Queues → New panel, perform the following actions. Scroll down to set all the following values:
  - Name: DonateQ
  - JNDI name: jms/DonateQ
  - Bus Name: ExperienceJEE
  - Queue Name: DonateQ
  - a. Click **OK**.

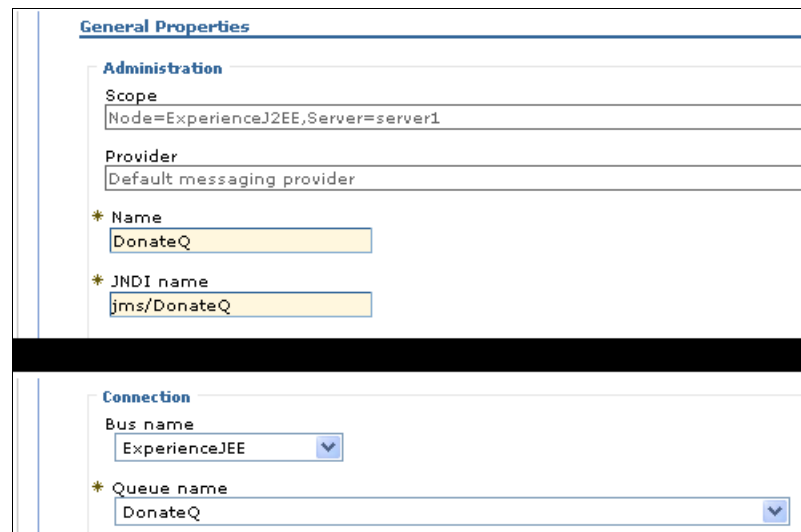
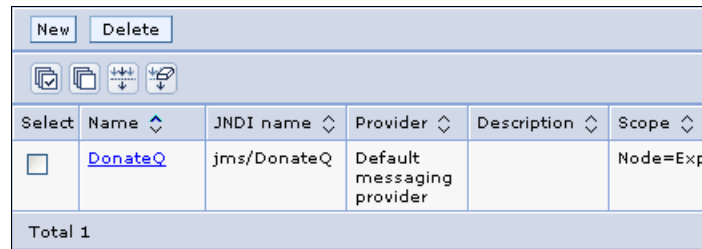


Figure 13-24 Define the JMS queue

6. Back at the Queues panel note the newly defined queue.



New Delete					
Select	Name	JNDI name	Provider	Description	Scope
<input type="checkbox"/>	DonateQ	jms/DonateQ	Default messaging provider		Node=Exp
Total 1					

Figure 13-25 New JMS queue

7. At the top of the panel, in the Messages box that states “Changes have been made to your local configuration...”, click **Save**.

### 13.4.5 Create the JMS activation specification

A JMS activation specification contains the configuration properties that control the run-time characteristics of a message-driven bean, such as which JMS queue is used, how many concurrent messages to process, and which user name to use when connecting to a secured messaging engine (assuming that security is enabled).

1. In the Admin Console, on the left select **Resources** → **JMS** → **Activation specifications**.
2. On the right, in the Activation specifications panel, select **Node=ExperienceJEE**, **Server = server1** from the scope pull-down.
3. On the right, in the refreshed Activation specifications panel, click **New**.
  - a. On the right, in the Activation specifications → Select JMS resource provider panel, accept the selection of **Default messaging provider**, and click **OK**.
  - b. On the right in the Activation specifications → Default messaging provider → New panel, configure the following settings. You may need to scroll down to see all of the following values:
    - Name: DonateAct
    - JNDI Name: jms/DonateAct
    - Destination type: Queue
    - Destination JNDI Name: jms/DonateQ
    - Bus name: ExperienceJEE
    - Authentication alias: ExperienceJEE/ExperienceJEEAlias
  - c. Click **OK**.

\* Name  
DonateAct

\* JNDI name  
jms/DonateAct

---

**Destination**

\* Destination type  
Queue

\* Destination JNDI name  
jms/DonateQ

Message selector

Bus name  
ExperienceJEE

---

**Additional**

Authentication alias  
ExperienceJEE/ExperienceJEEAlias

Figure 13-26 Activation specification properties

**Off course:** If you do not see the ExperienceJEEAlias repeat the steps in 4.6.2, “Create the JAAS authentication aliases” on page 103.

4. The Activation specification panel should reflect the new entry:

New Delete					
Select	Name	JNDI name	Provider	Description	Scope
<input type="checkbox"/>	DonateAct	jms/DonateAct	Default messaging provider		Node=Ex
Total 1					

Figure 13-27 New activation specification

5. At the top of the panel, in the Messages box that states “Changes have been made to your local configuration...”, click **Save**.

## 13.4.6 Create the JMS connection factory for the application client

A JMS connection factories provides a specific class of service that is used to connect to groups of queues and topics. If all your applications require the same class of service, you could use a single connection factory.

However, if your applications require different classes of service for example, one application requires that the inbound messages be processed as UNAUTHENTICATED while another application requires that inbound messages be processed as a specific user ID, then you need multiple connection factories.

Creating separate connection factories also allows you to distribute queues between multiple messaging providers (some local, some remote), enabling you to selectively shut down messaging for one application while leaving it active for another.

You use this connection factory with the messaging client that you create in 13.6, “Create the messaging client (DonateMDBClient)” on page 508.

Note that this connection factory is NOT required for the MDB that you create in 13.4, “Configure the test server for base JMS resources” on page 484. That artifact accesses the class of service through the activation specification.

Thus, you can defer this configuration step until 13.6, “Create the messaging client (DonateMDBClient)” on page 508, but it is done in this section to keep all the JMS configuration activities in one location.

1. In the Admin Console, on the left select **Resources** → **JMS** → **Connection factories**.
2. On the right, in the **Connection factories panel**, select **Node=ExperienceJEENode**, **Server = server1** from the scope pull-down.
3. On the right, in the refreshed Connection factories panel, click **New**.
  - a. In the Connection factories → Select JMS resource provider panel, accept the selection of **Default messaging provider**, and click **OK**.

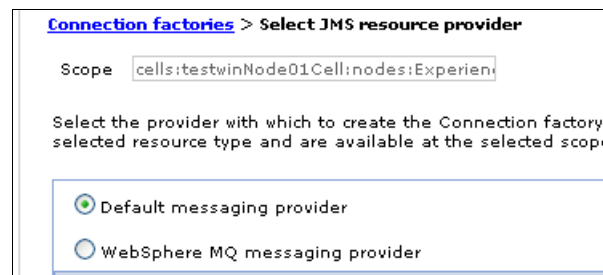


Figure 13-28 Select the messaging provider



- b. On the right, in the Connection factories → Default messaging provider → New panel, perform the following actions. Scroll down the panel to see all the following selections:
- Name: **DonateCF**
  - JNDI name: **jms/DonateCF**
  - Bus Name: **ExperienceJEE**
  - XA recovery authentication alias:  
**ExperienceJEENode/ExperienceJEEAlias**
  - Container-managed authentication alias:  
**ExperienceJEENode/ExperienceJEEAlias**
- c. Click **OK**.

#### **Behind the scenes:**

Java EE application clients cannot use the JAAS authentication alias information that is specified in the connection factory. Instead, they must supply this information using the `createConnection` method (as you do in 13.6, “Create the messaging client (DonateMDBClient)” on page 508).

However, earlier it was stated that this connection factory gets used in this chapter for the application client. Why configure the JAAS authentication alias if it cannot be used?

The answer is that in Chapter 14, “Add publication of results” on page 517, you update the `donateVacation` method (in the `Donate` session EJB) to publish the results to a JMS topic. That code requires a connection factory and JAAS authentication alias, and since it is running on the server, the code can use this information.

Therefore, you configure the JAAS authentication alias here to keep all the JMS configuration activities in one location.

4. The resulting Connection factories panel should reflect the new entry.





New Delete					
   					
Select	Name	JNDI name	Provider	Description	Scope
<input type="checkbox"/>	DonateCF	jms/DonateCF	Default messaging provider		Node=Experi
Total 1					

Figure 13-29 New connection factory

- At the top of the panel, in the Messages box that states “Changes have been made to your local configuration....”, click **Save**.
- Logout and close the **Admin Console**.

### 13.4.7 Restart the ExperienceJEE Test Server

Restart the test server because of the activation of the service integration bus and messaging engine.

- Switch to the Servers view in the lower-right pane, select **ExperienceJEE Test Server**, right-click and select **Restart**. The server status changes to Stopping, then Starting, and finally back to Started when the startup is complete.
- Switch to the Console view, and verify that the server started successfully. In particular look for the following message:

WSVR0001I: Server server1 open for e-business

## 13.5 Create the message-driven bean

In this section you use the standard workbench EJB tooling to create the message-driven bean (MDB).

### 13.5.1 Define the MDB

- In the Enterprise Explorer, select **DonateEJB**, right-click and select **New** → **Message-Driven Bean**.
- At the Create EJB 3.0 Message Driven Bean pop-up:
  - Ensure that EJB Project is set to **DonateEJB** and these settings.
    - Java package: **donate.mdb**

- Class name: **DonateMDB**
  - Leave **Destination name** blank and the **JMS selection** enabled.
  - Destination type: **Queue**
- b. Click **Next**.

Figure 13-30 Create the MDB

### Behind the scenes:

The Destination name field is used to populate the mappedName field in the MDB annotation, which points to the JMS JNDI queue name that will be used for this MDB.

However, putting this information into the class codes deployment time information into the source code. This information can be overridden at deployment time by a deployment descriptor but a general user might have difficulties in setting up that information.

Therefore, this book uses the approach of specifying the mapping in the deployment descriptor in the form of an activation specification. You will configure this in the next section (13.5.2, “Add the activation specification information” on page 504).

3. At the Create EJB 3.0 Message-Driven Bean / Enter Message-Driven Bean specification information pop-up, click **Next**.
4. At the Select Class Diagram for Visualization pop-up, clear the **Add Bean to Class Diagram** selection and click **Finish**.
5. Leave the resulting DonateMDB.java editor open because you will edit this in subsequent sections.

#### Behind the scenes:

The wizard generates a class that with the following MDB related information:

- The MessageDriven annotation:

```
@MessageDriven(  
    activationConfig = { @ActivationConfigProperty(  
        propertyName = "destinationType", propertyValue =  
        "javax.jms.Queue"  
    ) })
```

- A class definition that implements the MessageListener interface:

```
public class DonateMDB implements MessageListener {
```

- A default onMessage method (which will process received messages):

```
/**  
 * @see MessageListener#onMessage(Message)  
 */  
public void onMessage(Message message) {  
    // TODO Auto-generated method stub  
  
}
```

## 13.5.2 Add the activation specification information

In this section you update the MDB configuration to reference the same activation specification that you defined earlier in 13.4.5, “Create the JMS activation specification” on page 498.

1. In the Enterprise explorer, select **DonateEJB** (under DonateEJB), right-click and select **Java EE → Generate WebSphere Bindings Deployment Descriptor**.

**Behind the scenes:** this creates and opens `ibm-ejb-jar-bnd.xml` (in `/DonateEJB/ejbModule/META-INF`) which contains WebSphere Application Server specific binding information.

2. In the Enterprise explorer, select **DonateEJB2** (under DonateEJB), right-click and select **Java EE Generate WebSphere Bindings Deployment Descriptor**.
3. In the resulting EJB Bindings editor, switch to the design tab:

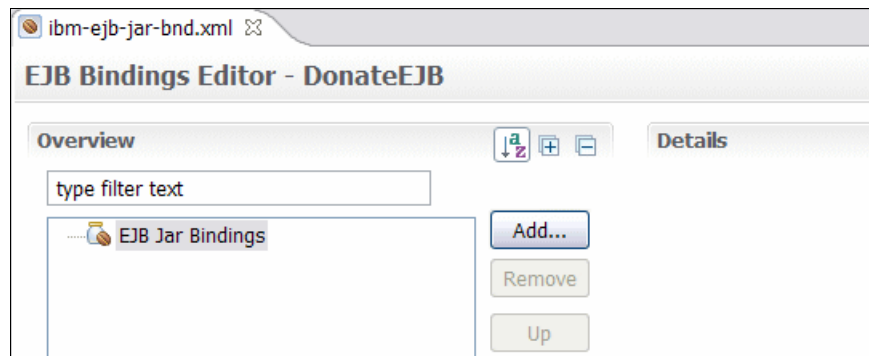


Figure 13-31 EJB bindings editor

- a. On the left select **EJB Jar Bindings** and click **Add**.
  - i. At the Add Item pop-up, select **Message Driven** and click **OK**.
  - ii. At the Message Driven Bean Binding pop-up, select **DonateMDB** and click **Finish**.
- b. Back at the EJB Bindings editor, on the left select **EJB Jar Bindings → Message Driven (DonateMDB)** and click **Add**.
  - i. At the Add item pop-up, select **JCA Adapter** and click **OK**.
- c. Back at the EJB Bindings editor:
  - i. On the left, ensure that **EJB Jar Bindings → Message Driven (DonateMDB) → JCA Adapter** is highlighted.
  - ii. On the right, next to Activation Spec Binding Name enter `jms/DonateAct`.
- d. Ensure that the editor is configured as show in Figure 13-32 on page 506.

- i. Save and close the EJB Bindings Editor (ibm-ejb-jar-bnd.xml).

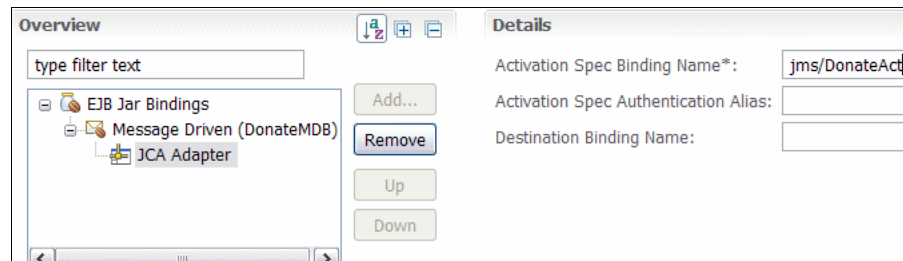


Figure 13-32 New bindings configuration

#### Behind the scenes:

This wizard adds the following stanza to `ibm-ejb-jar-bnd.xml` to bind the `DonateMDB` to the `jms/DonateAct` Activation Specification that you defined in 13.4.5, “Create the JMS activation specification” on page 498:

```
<message-driven name="DonateMDB">
  <jca-adapter activation-spec-binding-name="jms/DonateAct" />
</message-driven>
```

### 13.5.3 Code the `DonateMDB` message-driven bean

1. In the Enterprise Explorer, select **DonateMDB(.java)** (in `DonateEJB/ejbModule/donate.mdb`), and double-click to open the Java class.
2. In the resulting **DonateMDB.java** editor, insert the EJB annotation and the `handleMessage` method:
  - a. In the **DonateMDB.java** editor, insert a blank line before the closing brace, and position the cursor at this new line.
  - b. Insert the **F13-1: DonateMDB handleMessage** snippet from the Snippets view.
  - c. Organize imports (**Ctrl+Shift+O**) to resolve `javax.jms.TextMessage` and `donate.ejb.interfaces.DonateBeanRemote`. (Figure 13-1)

*Example 13-1 F13-1 DonateMDB handleMessage snippet*

```
@EJB
DonateBeanRemote donateBean;

private void handleMessage(Message msg) {
    try {
        String text = ((TextMessage) msg).getText();
```

```

        System.out.println("DonateMDB: Input Message = " + text);
        String[] tokens = text.split(" ");
        int employeeId = Integer.parseInt(tokens[0]);
        String fundName = tokens[1];
        int donationAmount = Integer.parseInt(tokens[2]);
        String result = donateBean.donateToFund(employeeId, fundName,
            donationAmount);
        System.out.println("DonateMDB: The Result Message is: " + result);
    } catch (Exception e) {
        System.out.println("DonateMDB: error in handleMessage: " +
            e.getLocalizedMessage());
    }
}

```

---

### Behind the scenes:

The @EJB annotation injects the reference to the DonateBean without using deployment descriptors. This capability was described in 7.3, “Define the Donate session EJB” on page 263.

The handleMessage method processes the input message as follows:

- ▶ Extract the text from the input message:
 

```
String text = ((TextMessage)msg).getText();
```
- ▶ Tokenize the text into the input parameters (employee number, fund key, donation amount):
 

```
String[] tokens = text.split(" ");
int employeeId = Integer.parseInt(tokens[0]);
String fundName = tokens[1];
int donationAmount = Integer.parseInt(tokens[2]);
```
- ▶ Invoke the donateVacation method, and print the result:
 

```
String result = donateBean.donateToFund(employeeId, fundName,
        donationAmount);
System.out.println("DonateMDB: The Result Message is: " +
    result);
```

3. In the **DonateMDB.java** editor, update the `onMessage` method to call the business logic **`handleMessage(msg)`** (created in the previous step) by including the highlighted statement:

```
/**
 * onMessage
 */
public void onMessage(javax.jms.Message message) {
    handleMessage(message);
}
```

4. Save and close `DonateMDB.java`.
5. Publish the `DonateEAR` enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
  - Switch to the **Console** view, and verify that `DonateEAR` restarted correctly after completing these changes by looking for:

```
WSVR0221I: Application started: DonateEAR
```

## 13.6 Create the messaging client (DonateMDBClient)

In this section, you create a Java EE application client that sends a JMS message to the `Donate` queue, thus allowing you to test the MDB.

1. Create the application client project:
  - a. In the Java EE Perspective Enterprise Explorer, select **DonateEAR**, right-click and select **New** → **Application Client Project**.
  - b. At the New Application Client Project/Application Client module pop-up:
    - i. Set the Project name to **DonateMDBClient**.
    - ii. Set Application Client module version to **5.0**.
    - iii. Set Configuration to **Default Configuration for WebSphere Application Server v7.0**.
    - iv. Under EAR membership, select **Add project to an EAR**, and select **DonateEAR**.
  - c. Click **Finish**.
2. In the Enterprise Explorer, select **Main.java** (in `DonateMDBClient/appClientModule/` (default package)) and double-click to open. Note that the class currently contains the following main method:

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
}
```



- a. In the **Main.java** editor, delete the existing main method and insert the **F13-2 DonateMDBCClient** snippet from the Snippets view.
- b. Organize imports (**Ctrl+Shift+O**) to resolve the following imports. Note that these class names exist under multiple packages, so ensure that you select the instances that match the imports show below:

```
import javax.annotation.Resource;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.naming.NamingException;
```

- c. Save and close Main.java.

```
@Resource(name = "DonateCF")
static ConnectionFactory cf;
@Resource(name = "DonateQ")
static Destination outDest;

public static void main(String[] args) throws NamingException,
JMSException {
    System.out.println("DonateMDBCClient: Establishing connection");
    Connection connection = cf.createConnection("javaeeadmin",
        "xxxxxxx");
    System.out.println("DonateMDBCClient: Establishing session");
    Session session = connection.createSession(false,
        Session.AUTO_ACKNOWLEDGE);
    MessageProducer destSender = session.createProducer(outDest);
    System.out.println("DonateMDBCClient: Sending message");
    TextMessage toutMessage = session.createTextMessage(args[0]);
    destSender.send(toutMessage);
    System.out.println("DonateMDBCClient: Message sent!");
    destSender.close();
    session.close();
    connection.close();
}
```

*Figure 13-33 F13-2 DonateMDBCClient snippet*

- d. Locate the following line:

```
Connection connection =
    cf.createConnection("javaeeadmin","xxxxxxx");
```

Change “xxxxxxx” to match the password you configured for javaeeadmin in 3.4, “Create the ExperienceJEE server profile” on page 62. For example, if the password is PASSWORD you would change this line to the following:

```
Connection connection =  
    cf.createConnection("javaeeadmin","PASSWORD");
```

e. Save and close Main.java.

### Behind the scenes:

The `@Resource` annotations automatically resolve the resource names to the connection factory and queue objects. The name used in the annotation is a reference, and this reference is mapped to a JNDI name in the Java EE binding file (which you will create in the next step).

It is also possible to get the connection factory and queue without annotation by programmatically retrieving it from the context.

```
Context ctx = new InitialContext();
ConnectionFactory cf =
    (ConnectionFactory) ctx.lookup("jms/DonateCF");
Destination outDest = (Destination)
    ctx.lookup("jms/DonateQ");
```

The main method performs the following:

- ▶ Create and start the connection:

```
Connection connection =
    cf.createConnection("javaeeadmin", "xxxxxxx");
```

- ▶ Create the session:

```
Session session =
    connection.createSession(false,
    Session.AUTO_ACKNOWLEDGE);
```

- ▶ Create the destination sender:

```
MessageProducer destSender =
    session.createProducer(outDest);
```

- ▶ Create and set the message:

```
TextMessage toutMessage =
    session.createTextMessage(args[0]);
```

- ▶ Send the message:

```
destSender.send(toutMessage);
```

- ▶ Free up resources:

```
destSender.close();
session.close();
connection.close();
```

3. Add the reference binding to the deployment descriptor.

- a. In the Enterprise Explorer view, select **DonateMDBClient**, right-click and select **Java EE → Generate WebSphere Bindings Deployment Descriptor**.
- b. In the resulting Application Client Bindings editor:
  - i. Switch to the Design tab
  - ii. Select **Application Client Bindings** and click **Add**.
  - iii. At the Add item pop-up, select **Message Destination Reference** and click **OK**.
- c. Back at the Application Client Bindings editor:
  - i. On the left, select **Application Client Bindings → Message Destination Reference**.
  - ii. On the right, next to name enter **DonateCF**.
  - iii. On the right, next to Binding Name enter **jms/DonateCF**.
- d. On the left **Application Client Bindings** and click **Add**.
  - i. At the Add item pop-up, select **Message Destination Reference** and click **OK** to add a second Message Destination Reference.
- e. Back at the Application Client Bindings editor:
  - i. On the left, select the second **Message Destination Reference** under Application Client Bindings.
  - ii. On the right, next to name enter **DonateQ**.
  - iii. On the right, next to Binding Name enter **jms/DonateQ**.
- a. Verify that the Application Client Bindings editor displays the following configuration.
  - i. Save and close the Application Client Bindings editor (ibm-application-client-bnd.xml).

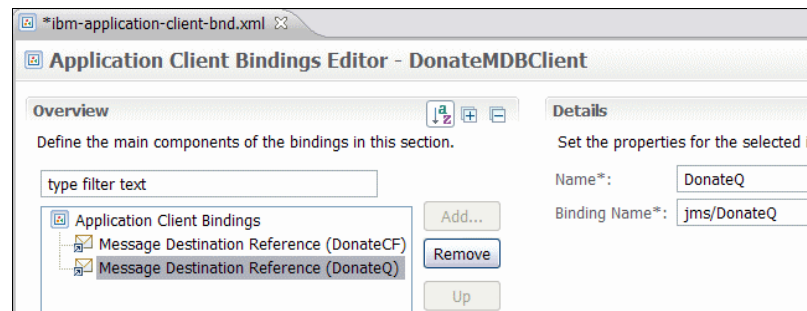


Figure 13-34 Application client bindings configuration

**Behind the scenes:** the editor inserts the following stanzas into `ibm-application-client-bnd.xml`:

```
<message-destination-ref name="DonateCF"
    binding-name="jms/DonateCF" />
<message-destination-ref name="DonateQ"
    binding-name="jms/DonateQ" />
```

### Behind the scenes:

You have to provide a user ID and password when creating a connection because you enabled the SIB to run in a secure mode (in 13.4.1, “Create the Service Integration Bus” on page 484).

If you were running a server-side artifact (either in a Web container or in an EJB container), this information can come from a JAAS authentication alias listed in the connection factory or the activation specification.

Client-side programs cannot access this information. Therefore, the option used here is to provide the user ID and password on the `createConnection` call.

- ▶ If you supplied no ID the `createConnection` operation fails with multiple messages, including the following:  
  
CCWSIT0094E: The client was not authorized to connect to bus ExperienceJEE using the bootstrap server with endpoint `localhost:7276:BootstrapBasicMessaging`. Reason: CWSIT0109E: The application did not specify a user ID when attempting authorization with the bus ExperienceJEE.
- ▶ If you supply an invalid ID the `createConnection` operation fails with multiple messages, including the following:  
  
CWSIA0004E: The authentication for the supplied user name `javaeeadminx` and the associated password was not successful.

Per the setup in 13.4.2, “Configure security” on page 490, you must use one of the following user IDs on the `createConnection` operation:

- The `javaeeadmin` user ID.
- Any user that belongs to the `DUSERS` group (`DNARMSTRONG` or `DCBROWN`).

## 13.7 Test the messaging client

The following steps are similar to the those used in 9.6, “Test the application client” on page 350 and 10.7.3, “Update and test the Java EE application client test configuration” on page 403.


1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. In the Enterprise Explorer, select **DonateMDBClient**, right-click and select **Run As → Run Configurations...**
3. At the Create, Manage, and run Configurations pop-up:
  - a. On the left, under Configurations, select **WebSphere v7.0 Application Client**, right-click and select **New**.
  - b. On the resulting right pane set Name to **DonateMDBClient**.
  - c. On the right pane Application tab set these values:
    - WebSphere Runtime: WebSphere Application Server V7
    - Enterprise application: DonateEAR
    - Application client module: DonateMDBClient
  - i. Select **Enable application client to connect to a server**.
  - ii. Select **Use specific server**, and then select **ExperienceJEE Test Server**
  - d. On the right pane Arguments tab:
    - i. Append “**1 DonationFund 1**” to the program argument string, making sure that you leave a space between it and the existing arguments and that you surround the new arguments with double quotes. The resulting program arguments string should be similar to the following:  
`-CCverbose=true “1 DonationFund 1”`
  - e. Click **Apply** to save the configuration (the pop-up remains open).
  - f. Click **Run** to execute the configuration (and the pop-up closes).
4. At the Login at the Target Server pop-up, click **Cancel**.

**Behind the scenes:** the application client launch configuration automatically queries for JAAS credentials, but DonateMDBClient does not require any credentials.

If you do not see this pop-up, minimize the workbench and other windows to ensure that this is not obscured.

5. Switch to the Console view to observe the execution results

```
DonateMDBClient: Establishing connection
DonateMDBClient: Establishing session
DonateMDBClient: Sending message
DonateMDBClient: Message sent!
```

6. Switch the **Console** view to the ExperienceJEE Test Server output using the Console selection icon [] from the view action bar, and verify that the message processed successfully (Note there may be other messages between these):

```
SystemOut      0 DonateMDB: Input Message = 1 DonationFund 1
SystemOut      0 EmployeeFacade:
em=JPATxEntityManager@195e195e[PuId=DonateEAR#DonateEJB.jar#DonateJP
AEmployee,
DonateEAR#DonateEJB.jar#EmployeeFacade#vacation.ejb.impl.EmployeeFac
ade/em]
SystemOut      0 DonateMDB: The Result Message is: Transfer successful
```

## 13.8 Explore!

This IBM Redbook implements messaging using the embedded JMS provider shipped with WebSphere Application Server. This implementation provides a robust implementation for JMS to WebSphere Application Server interaction, but it is not capable of supporting other environments, such as non-Java or non-JMS applications.

### 13.8.1 Integration with WebSphere MQ

Note that you can configure WebSphere Application Server to use either the embedded JMS provider or WebSphere MQ for MDBs. Thus, you can implement this chapter using WebSphere MQ with a few changes, most notably to configure WebSphere MQ resources (instead of Default messaging provider resources) and to implement the EJB using the listener port artifact instead of the activation specification (because the activation specification requires JCA 1.5 support, and WebSphere MQ currently does not have that support).

Additional explore resources are available at the following Web addresses:

- ▶ WebSphere MQ product page:  
<http://www.ibm.com/software/integration/wmq>
- ▶ WebSphere Application Server InfoCenter: Interoperation with WebSphere MQ:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/cmm\\_interop.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/cmm_interop.html)





## Add publication of results

In this chapter, we upgrade the message-driven bean application by publishing the results to a JMS topic from the session bean. We also create a Java EE application client to subscribe to messages sent to this topic.

## 14.1 Learn!

Figure 14-1 shows the application publication results.

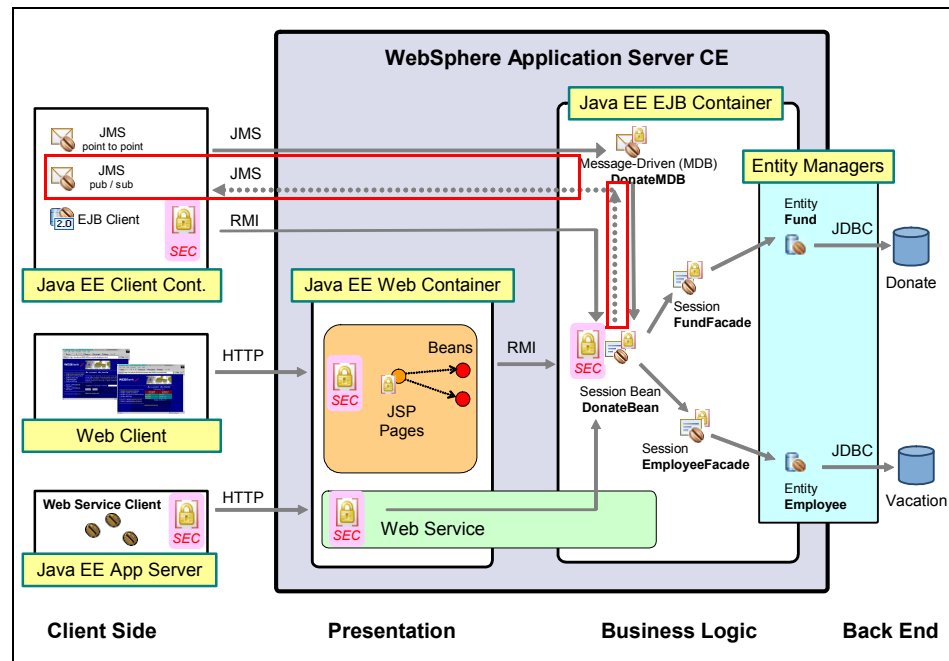


Figure 14-1 Donate application: Publication of results

The point-to-point messaging used in the previous chapter assumes that the message is sent to a single known destination. What if you do not know the intended destination? What if multiple programs might require this message?

This type of messaging need can be addressed by using a topic that represents a special type of messaging destination. It is similar to a subscription service to an electronic bulletin board, and has two important characteristics:

- ▶ A name, which can be a single-level name (such as employee numbers 1 or 2 used in this scenario) or hierarchical (/stocks/IBM or /computers/IBM). Topics can be accessed using wildcards (/stocks/\* or even \*/IBM).
- ▶ A subscription list, representing destinations that want to receive the messages sent to this topic.

A topic can be a pre-existing resource accessed through the JNDI name or it can be dynamically defined:

- **Publisher:** A program creates a message and sends it to a JMS topic.
- **Subscriber:** A program connects to a specific JMS topic, and then issues a receive. When the publish and subscribe engine receives a message for that topic, it places a copy of the message on the queue for each open receive (Figure 14-2).

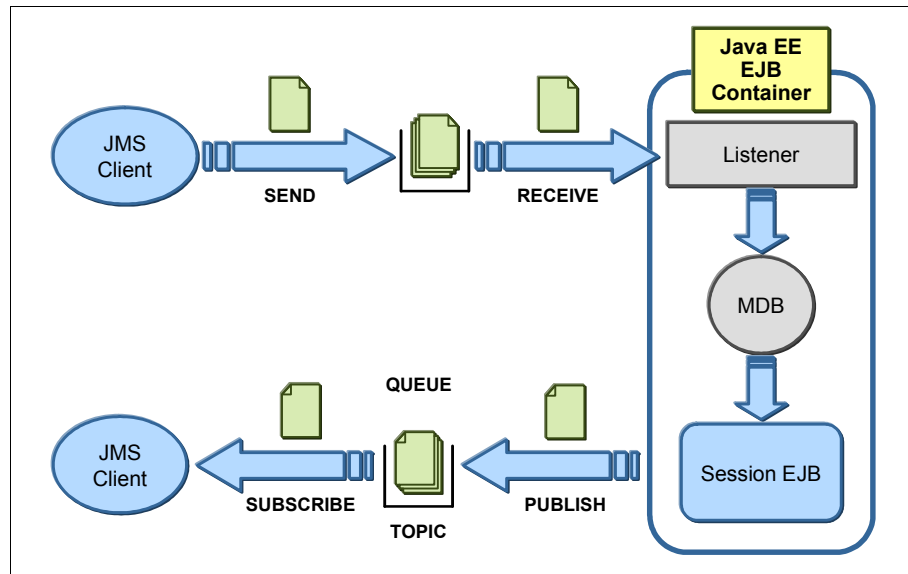


Figure 14-2 JMS with publisher and subscriber

There are no unique Learn resources for this chapter.

## 14.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed all the applications of Part 2, “Core Java EE application” on page 121. In addition, you must have completed Chapter 13, “Create the message-driven bean” on page 477.

## 14.3 Update the DonateBean session EJB

In this section, we update the DonateBean session EJB to publish the results of a donateToFund invocation to a topic:

1. In the Enterprise Explorer, open **DonateBean** (in DonateEJB/ejbModule/donate.ejb.impl).
2. Add the required local variables for pub/sub processing, after the @EJB annotations, and before the donateToFund method:

```
/** Added for pub sub logic  
  
@Resource(name = "DonateCF")  
ConnectionFactory cf;  
Connection connection;  
Session session;
```

- a. Organize imports (**Ctrl+Shift+O**) to resolve:

```
javax.jms.Connection  
javax.jms.ConnectionFactory  
javax.jms.Session
```

### Behind the scenes:

What makes EJB development testing difficult are EJB code dependencies on data sources and JMS resources, as well as how EJB components are invoked by EJB clients. The EJB 3.0 specification introduces dependency injection as a mechanism to alleviate these difficulties. Instead of using JNDI lookups, an EJB can define a resource reference by injecting code. Here is an example of an EJB bean that has to call another EJB and use a data source to perform JDBC work:

```
@EJB
DonateBeanRemote donateBean;

@Resource (name="jdbc/donatedb")
private DataSource ds;
```

Note that the resource name is not a JNDI name: it is a reference that must be bound in the application deployment descriptor binding file.

Dependency injection can occur in a variety of ways, for example, via a setter method or a class variable. See the specification for more details:

<http://www.jcp.org/en/jsr/detail?id=220>

This reference is taken from “Examining the EJB 3.0 Simplified API Specification,” available at:

[http://www.ibm.com/developerworks/websphere/techjournal/0502\\_col\\_barciarcia/0502\\_col\\_barciarcia.html](http://www.ibm.com/developerworks/websphere/techjournal/0502_col_barciarcia/0502_col_barciarcia.html)

3. Add a `createSession` method before the closing brace by inserting the **F14-1 MDB PubSub Create Session** snippet (Example 14-1).

- a. Organize imports (**Ctrl+Shift+O**) to resolve:

```
import javax.annotation.PostConstruct;
import javax.ejb.EJBException;
```

#### *Example 14-1 F14-1 MDB PubSub createSession*

```
// Added for pub/sub logic
@PostConstruct
public void createSession() {
    try {
        connection = cf.createConnection();
        session = connection.createSession(false,
            Session.AUTO_ACKNOWLEDGE);
        connection.start();
    }
```

```

        } catch (Exception e) {
            throw new EJBException("Donate: error in pub/sub createSession",
e);
        }
    }
}

```

---

#### Behind the scenes:

Before EJB 3.0 and annotations, developers had to use life cycle methods such as `ejbCreate`. Now, with annotations, it has become much easier. The EJB developer can simply use the `@PostConstruct` annotation before a method and this method is executed following the dependency injection. This is very useful for executing class initialization code.

4. Add a `closeSession` method before the closing brace by inserting the **F14-2 MDB PubSub Close Session** snippet (Example 14-2).
  - a. Organize imports (**Ctrl+Shift+O**) to resolve:

```

import javax.annotation.PreDestroy;
import javax.jms.JMSException;

```

#### Example 14-2 F14-2 MDB PubSub Close Session

---

```

// Added for pub/sub logic
@PreDestroy
public void closeSession() {
    try {
        if (session != null) session.close();
    } catch (JMSException e) { e.printStackTrace(); }
    try {
        if (connection != null) connection.close();
    } catch (JMSException e) { e.printStackTrace(); }
}

```

---

#### Behind the scenes:

Similar to `@PostConstruct`, the EJB developer can use the `@PreDestroy` annotation to release the resources associated with the session and connection when the EJB is removed.

5. Add the `publishResults` method before the closing brace by inserting the **F4-3 MDB PubSub Publish Results** snippet (Example 14-3 on page 523).
  - a. Organize imports (**Ctrl+Shift+O**) to resolve:

```

import javax.jms.Destination;

```

```
import javax.jms.MessageProducer;
import javax.jms.TextMessage;
```

#### *Example 14-3 F14-3 MDB PubSub Publish Results*

---

```
// Added for pub/sub logic
private void publishResults(int employeeId, String fundName, int hours,
String return_string) {
    try {
        Destination empTopic = session
            .createTopic(String.valueOf(employeeId));
        MessageProducer empProducer = session.createProducer(empTopic);
        TextMessage pubMessage = session.createTextMessage("Request = "
            + employeeId + " " + fundName + " " + hours + " , result = "
            + return_string);
        empProducer.send(pubMessage);
        System.out.println("Message sent to " + empTopic + " : "
            + pubMessage.getText());
        empProducer.close();
    } catch (Exception e) {
        System.out.println("Error in pub/sub publishResults: "
            + e.getLocalizedMessage());
    }
}
```

---

#### **Behind the scenes:**

This code performs the following functions:

- Create a (topic) destination for the employee:

```
Destination empTopic =
    session.createTopic(String.valueOf(employeeId));
```

- Create a producer for the destination:

```
MessageProducer empProducer = session.createProducer(empTopic);
```

- Create the message:

```
TextMessage pubMessage = session.createTextMessage(...);
```

- Publish the message:

```
empProducer.send(pubMessage);
```

- Free up resources:

```
empProducer.close();
```

6. Update the second `donateToFund` method (with simple arguments) to call the `publishResults` method:

```
public String donateToFund(int employeeId, String fundName, int hours) {
    String return_string = null;
    try {
        .....
    }
    publishResults(employeeId, fundName, hours, return_string);
    return return_string;
}
```

7. Organize imports (**Ctrl+Shift+O**) and select the following import:

```
javax.jms.Destination
```

8. Save and close `DonateBean.java`.

9. Add the reference binding to the deployment descriptor.

- a. In the Enterprise Explorer view, open **ibm-ejb-jar-bnd.xml** (in `DonateEJB/ejbModule/META-INF`)
- b. In the resulting EJB Bindings editor, switch to the **Design** tab.
- c. On the left select **EJB Jar Bindings** and click **Add**.
  - i. At the Add Item pop-up, select **Session** and click **OK**.
  - ii. At the Session Bean Binding / Select the session bean ... pop-up, select **DonateBean** and click **Next**.
  - iii. At the Session Bean Binding / Select the type of session bean binding .. pop-up, select **Resource Reference** and click **Next**.
  - iv. At the Session Bean Binding / Select the resource reference ... pop-up, select **DonateCF** and click **Next**.
  - v. At the Session Bean Binding / Enter the binding name ... pop-up, enter **jms/DonateCF** and click finish.
- d. Save and close `ibm-ejb-jar-bnd.xml`.

**Behind the scenes:** this wizard added the following standards to the ejb bindings deployment descriptor:

```
<session name="DonateBean">
    <resource-ref name="DonateCF" binding-name="jms/DonateCF">
    </resource-ref>
</session>
```

10. Publish the `DonateEAR` enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).



## 14.4 Create the messaging pub sub client (DonatePubSubClient)

These steps are similar to those used in 13.6, “Create the messaging client (DonateMDBClient)” on page 508.

1. Create the application client project:
  - a. In the Java EE Perspective Enterprise Explorer, select **DonateEAR**, right-click and select **New** → **Application Client Project**.
  - b. At the New Application Client Project/Application Client module pop-up:
    - i. Set the Project name to **DonatePubSubClient**.
    - ii. Set Application Client module version to **5.0**.
    - iii. Set Configuration to **Default Configuration for WebSphere Application Server v7.0**.
    - iv. Under EAR membership, select **Add project to an EAR**, and select **DonateEAR**.
    - v. Click **Finish**.
2. In the Enterprise Explorer, select **Main.java** (in DonatePubSubClient/appClientModule/ (default package)) and double-click to open. Note that the class currently contains the following main method:

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
}
```

- a. In the **Main.java** editor, delete the existing main method and insert the **F14-4: MDB PubSub Client Main** snippet from the Snippets view.
- b. Organize imports (**Ctrl+Shift+O**) to resolve the following imports. Note that these class names exist under multiple packages, so ensure that you select the instances that match the imports show below:

```
import javax.annotation.Resource;  
import javax.jms.Connection;  
import javax.jms.ConnectionFactory;  
import javax.jms.Destination;  
import javax.jms.JMSException;  
import javax.jms.MessageConsumer;  
import javax.jms.Session;  
import javax.jms.TextMessage;  
import javax.naming.NamingException;
```

*Example 14-4 F14-4: MDB PubSub Client Main*

---

```
@Resource(name = "DonateCF")
```

```

static ConnectionFactory cf;

public static void main(String[] args) throws NamingException,
JMSException {
    System.out.println("DonatePubSubClient: Establishing connection");
    Connection connection = cf.createConnection("javaeeadmin",
"xxxxxxx");
    connection.start();

    System.out.println("DonatePubSubClient: Establishing session");
    Session session = connection.createSession(false,
        Session.AUTO_ACKNOWLEDGE);
    Destination empTopic = session.createTopic(args[0]);
    MessageConsumer empSub = session.createConsumer(empTopic);

    System.out
        .println("DonatePubSubClient: Receiving messages for updates to
employee "
            + empTopic.toString());
    try {
        while (true) {
            System.out.println("returned message = "
                + ((TextMessage) empSub.receive()).getText());
        }
    } catch (Exception e) {

        // * closing logic
        System.out.println("exit exception block: " + e.getMessage());
        empSub.close();
        session.close();
        connection.close();
    }
}connection.close();

```

---

c. Locate the following line:

```

Connection connection =
    cf.createConnection("javaeeadmin", "xxxxxxx");

```

Change “xxxxxxx” to match the password you configured for javaeeadmin in 3.4, “Create the ExperienceJEE server profile” on page 62. For example, if the password is PASSWORD you would change this line to the following:

```

Connection connection =
    cf.createConnection("javaeeadmin", "PASSWORD");

```

d. Save and close Main.java.

**Behind the scenes:** the example above performs the following actions:

- ▶ Create/Start the connection:

```
Connection connection = cf.createConnection("javaeadmin",
   "xxxxxxx");
connection.start();
```
- ▶ Create the session:

```
Session session =
    connection.createSession(false,
    Session.AUTO_ACKNOWLEDGE);
```
- ▶ Create the (topic) destination:

```
Destination empTopic = session.createTopic(args[0]);
```
- ▶ Create the consumer:

```
MessageConsumer empSub = session.createConsumer(empTopic);
```
- ▶ Loop forever on receiving messages:

```
empSub.receive()).getText();
```
- ▶ If the code is ever modified to NOT loop forever, free resources:

```
empSub.close();
session.close();
connection.close();
```

3. Add the reference binding to the deployment descriptor.
  - a. In the Enterprise Explorer view, select **DonatePubSubClient**, right-click and select **Java EE → Generate WebSphere Bindings Deployment Descriptor**.
  - b. In the resulting Application Client Bindings editor:
    - i. Switch to the Design tab
    - ii. Select **Application Client Bindings** and click **Add**.
    - iii. At the Add item pop-up, select **Message Destination Reference** and click **OK**.
  - c. Back at the Application Client Bindings editor:
    - i. On the left, select **Application Client Bindings → Message Destination Reference**.
    - ii. On the right, next to name enter **DonateCF**.
    - iii. On the right, next to Binding Name enter **jms/DonateCF**.

- d. Save and close the Application Client Bindings editor (ibm-application-client-bnd.xml)

## 14.5 Test the messaging pub sub client






DonatePubSubClient uses non-durable subscriptions, which means that it only receives messages published to the topic after it is started.

Therefore, in this section you first start DonatePubSubClient, and then submit requests through DonateMDBClient and the Web interface.

JMS supports the concept of durable subscriptions, which allows a program to receive messages that were sent to the topic while it was off-line. Reference 14.6, “Explore!” on page 531 for information about both durable subscriptions and a related topic of message persistence.

### 14.5.1 Start DonatePubSubClient


The following steps are similar to the steps used in 13.7, “Test the messaging client” on page 514.

1. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to
    -  `C:\IBM\SDP\runtimes\base_v7\derby\bin\networkServer`
    -  `/opt/IBM/SDP/runtimes/base_v7/derby/bin/networkServer`
  - and execute:
    -  `startNetworkServer.bat`
    -  `startNetworkServers.sh`
  - If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
    - If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.
2. In the Enterprise Explorer, select **DonatePubSubClient**, and from the workbench action bar select **Run** → **Run...**
3. At the Create, Manage, and run Configurations pop-up:

- a. On the left, under Configurations, select **WebSphere v7.0 Application Client**, right-click and select **New**.
  - b. On the resulting right pane set Name to **DonatePubSubClient**.
  - c. On the right pane Application tab set these values:
    - WebSphere Runtime: WebSphere Application Server V7
    - Enterprise application: DonateEAR
    - Application client module: DonatePubSubClient
    - i. Select **Enable application client to connect to a server**.
    - ii. Select **Use specific server**, and then select **ExperienceJEE Test Server**
  - d. On the right pane Arguments tab:
    - i. Append **1** to the program argument string. The resulting program arguments string should be similar to the following:  
`-CCverbose=true 1`
  - e. Click **Apply** to save the configuration (the pop-up remains open).
  - f. Click **Run** to execute the configuration (and the pop-up closes).
4. At the Login at the Target Server pop-up, click **Cancel**.

**Behind the scenes:** the application client launch configuration automatically queries for JAAS credentials, but DonateMDBClient does not require any credentials.

If you do not see this pop-up, minimize the workbench and other windows to ensure that this is not obscured.


5. Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [  ] from the view action bar, and then verify that the Application Client starts successfully and is waiting for messages.

```
DonatePubSubClient: Establishing initial context
DonatePubSubClient: Looking up ConnectionFactory
DonatePubSubClient: Establishing connection
DonatePubSubClient: Establishing session
DonatePubSubClient: Receiving messages for updates to employee
topic://1
```


## 14.5.2 Submit requests

Any request that invokes the Donate session EJB—whether started through an MDB, a Web browser, or a Web service—causes a message to be published to the topic.


1. Submit a request for employee 1 through the DonateMDBClient by performing the following actions:

- a. In the workbench action bar select **Run** → **Run...**
- b. On the left, select **WebSphere v7 Application Client** → **DonateMDBClient**, and in the lower right click **Run**. This submits a message for employee 1 for DonationFund for 1 day.
- c. Switch the **Console** view to the **DonateMDBClient** output using the Console selection icon [  ] from the view action bar, and then verify that it sent a message.

```
DonateMDBClient: Establishing connection
DonateMDBClient: Establishing session
DonateMDBClient: Sending message
DonateMDBClient: Message sent!
```

- d. Switch the **Console** view to the ExperienceJEE Test Server output using the Console selection icon [  ] from the view action bar, and then verify that the MDB received the message and called donateVacation (and that donateVacation published the results).


```
DonateMDB: Input Message = 1 DonationFund 1
EmployeeFacade: User id=UNAUTHENTICATED
EmployeeFacade: isCallerInRole=false
Message sent to topic://1 : Request = 1 DonationFund 1 , result =
null
DonateMDB: The Result Message is: Transfer successful
```


- e. Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [  ] from the view action bar, and then verify that it received the result message.



```
...
...
```

```
DonatePubSubClient: Establishing session
DonatePubSubClient: Receiving messages for updates to employee
topic://1
returned message = Request = 1 DonationFund 1 , result = Transfer
Successful
```

2. Submit a request for employee 1 through the Web interface by performing the following actions.

- a. In the Enterprise Explorer, select **FindEmployee.jsp** (in /DonateJSFWeb/WebContent), right-click and select **Run As** → **Run on Server...**
  - b. In the FindEmployee.jsp page, enter an EmployeeID of **1**, and click **Submit**.
  - c. In the **EmployeeDetail.jsp** page, enter a Donate hours value of **1** (or any value), and click **Submit**.
  - d. Switch the **Console** view to the ExperienceJEE Test Server output using the Console selection icon [  ] from the view action bar, and then verify that the MDB received the message and called donateVacation (and that donateVacation published the results).
 

```
SystemOut      0 Message sent to topic://1 : Request = 1
DonationFund 1 , result = Transfer Successful
```
  - e. Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [  ] from the view action bar, and then verify that it received the result message.
 

```
returned message = Request = 1 DonationFund 1 , result = Transfer
Successful
```
  - f. Close the Web browser.
3. Submit a request for employee 2 through the Web interface by performing the same actions above. following actions.
    - a. Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [  ] from the view action bar, and then verify that it did NOT receive the result message because the **DonatePubSubClient** is subscribing to updates for employee 1, NOT for employee 2.
    - b. Close the Web browser.
  4. Terminate the **DonatePubSubClient**.
    - a. Switch to the **DonatePubSubClient** output in the **Console** view, and click the terminate icon [  ].

## 14.6 Explore!

Section 13.8, “Explore!” on page 515 discussed how WebSphere Application Server can utilize WebSphere MQ instead of the embedded JMS provider for base messaging transport. In a similar manner, WebSphere Application Server can utilize an external product for publish and subscribe capability, in particular the WebSphere Event Broker and the WebSphere Message Broker.

These products provide enhanced monitoring and configuration tools for publish and subscribe, as well as additional capabilities that can transform and route the messages.

JMS messages are maintained asynchronously from the producer and the consumer. Therefore, you need to consider how the messages should be managed, meaning how they are preserved (or not).

There are two concepts that apply:

- ▶ Durability, which refers to what happens to topic messages when the topic subscriber is off-line. JMS has two different API calls that determine if a durable or non-durable subscription is used (`session.createDurableSubscriber` and `session.createConsumer`).

The default that we use in this document is non-durable subscriptions, which indicates that the subscriber only receives messages (that match that subscription) that arrive while the subscriber is active (after a `session.CreateConsumer` call is issued). If the subscriber disconnects and then reconnects later, it will not receive messages that were published while it was disconnected.

- Advantages: Simpler programming model for the program subscribing to the messages, and less overhead on the server side.
- Disadvantages: The consumer does not receive all messages sent to the topic over time.

Durable subscriptions receive all messages (that match the subscription) regardless of whether the client was active when the message was sent.

- Advantages: The client receives all messages that match the topic.
- Disadvantages: The publish and subscribe server needs to maintain additional resources to store the messages and subscription information.

Durability is driven purely by the client program, which decides whether to access a topic as durable or non-durable.

- ▶ Persistence, which refers to how messages are preserved relative to the overall availability of the infrastructure. This applies to both queues and topics.

Non-persistent indicates that messages are discarded when the messaging engine stops, fails, or was unable to obtain sufficient system resources. With this configuration, the messages are effectively maintained in a memory store.

- Advantages: Messages are processed with a minimum of overhead because there is no file access.
- Disadvantages: The messages may be lost.



Persistent (the default value used in this book) indicates that messages are preserved. This is sometimes called “assured delivery”: The infrastructure guarantees that the message is available on the destination queue, but obviously can not guarantee if or when a user program actually reads the message.

- Advantages: All messages are preserved because they are written to a file store. WebSphere Application Server can be configured to use a centralized file store across many servers, allowing fail-over and workload management.
- Disadvantages: Messages are processed with an additional overhead because there is file access.

Persistence is determined by a hierarchy of settings:

- The SIB destination definition, for example queue, sets the “default reliability” (Best effort nonpersistent, Express nonpersistent, Reliable nonpersistent, Reliable persistent, Assured persistent) and can choose to fix this value and deny others the ability to override.
- The JMS connection factory definition can set the “quality of service” characteristics both for persistent and non-persistent access. One of the options is *As bus destination*, which indicates that the access should default to that set in the SIB destination.
- The JMS queue definition sets the “delivery mode” mode to persistent, non-persistent, or lets the client program decide.
- The client program can set the delivery mode to persistent, non-persistent, or use the default delivery mode by using the `setDeliveryMode` method on the Message Producer object.

Additional explore resources are available at the following Web sites:

- ▶ WebSphere Application Server InfoCenter: Publishing and subscribing with a WebSphere MQ network:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.pmc.doc/concepts/cjc0005\\_.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.pmc.doc/concepts/cjc0005_.html)
- ▶ WebSphere Application Server InfoCenter: Using durable subscriptions:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.pmc.doc/tasks/tjn0012\\_.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.pmc.doc/tasks/tjn0012_.html)
- ▶ WebSphere Application Server InfoCenter: Learning about file stores:  
[http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.pmc.doc/tasks/tjm2000\\_.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.pmc.doc/tasks/tjm2000_.html)





## Implement security for messaging

In this chapter, we update the MDB (DonateMDB) and the Java EE application client (DonateMDBClient) to operate with Java EE application security enabled.

## 15.1 Learn!

Figure 15-1 shows the updated MDB (DonateMDB) and the Java EE application client (DonateMDBClient) to operate with Java EE application with security enabled.

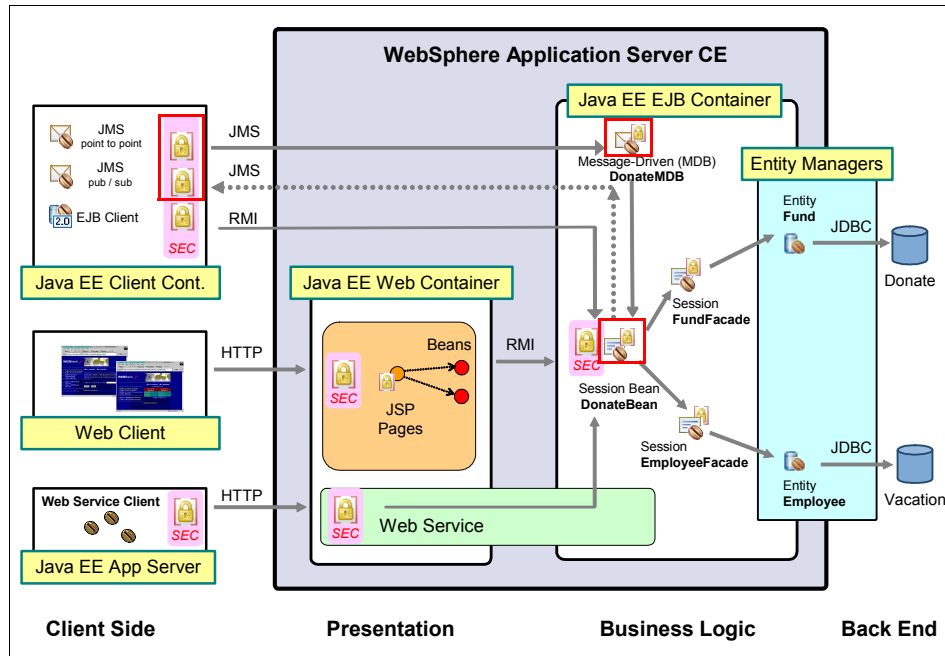


Figure 15-1 Donate application: Message security

Messaging has two levels of security:

- ▶ The first level controls interaction with the messaging provider to send and receive messages, for example, when we send JMS messages from a client, as in 13.6, “Create the messaging client (DonateMDBClient)” on page 508. We will update the client code with user ID and password information.
- ▶ The second level is setting the user context for the message received by an MDB. When an MDB receives a message through the `onMessage` method, which user ID should this message be executed under? This is important if the `onMessage` method then invokes a session EJB that is secured by either Java EE declarative security or programmatic security.

For the second level, Java EE provides the following two choices:

- ▶ Run without credentials (default), meaning that the `onMessage` method and any subsequent method or EJB in the calling chain runs as `UNAUTHENTICATED`.

What happens if one of the EJBs requires a credential, for example, if role based security is applied to the EJB? The answer is that the access fails.

- ▶ Run as a specific user called a *run-as-identity*, defined through an entry in the EJB deployment descriptor, which points to a JAAS Authentication Alias (defined in the Java EE runtime environment) containing the user ID and password. The *DonateMDB* runs successfully without code changes if you add a *run-as-identity*.

Both of these Java EE alternatives have drawbacks:

- ▶ Running without client credentials means that you cannot access secured EJBs.
- ▶ Running with a *run-as-identity* means that all invocation of secured EJBs are done in the context of the same user, thus minimizing the value of implementing role-based security.

This scenario implements a third alternative: The message sender provides a user ID and password in the JMS properties header contained in the message. The *onMessage* method uses these to create a login context, and then invokes the *handleMessage* method using that login context.

The advantage of this approach (over the two Java EE provided alternatives) is that you maintain the integrity of the Java EE role-based security. The drawback is that users accessing the program must supply a user ID password in the JMS properties.

Alternatively, you can create a security realm with a custom login module, pass any information in the JMS header, extract the header, and pass information into a callback handler.

There are no unique Learn resources for this chapter. Instead, refer to the security specific sections in the Learn resources referenced in 13.1, “Learn!” on page 478.

## 15.2 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed all the applications of Part 2, “Core Java EE application” on page 121. In addition, you must have finished Chapter 14, “Add publication of results” on page 517.

## 15.3 Re-enable Java EE EJB declarative security

In this section we perform the set of tasks required to re-enable the Java EE EJB declarative security that we disabled in 13.3, “Disable Java EE EJB declarative security” on page 483:

1. Re-enable declarative security in the DonateBean session EJB:

- a. In the Enterprise Explorer, open **DonateBean** (in DonateEJB/ejbModule/donate.ejb.impl), and switch the security annotations for both donateToFund methods (remove the comment marker from @RolesAllowed and add a comment marker to @PermitAll):

```
@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
//@PermitAll
public void donateToFund(Employee employee, Fund fund, int hours)
.....



@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
//@PermitAll
@WebMethod
public String donateToFund(int employeeId, String fundName, ...)
.....
```

- b. Organize the imports (**Ctrl+Shift+O**).



- c. Save and close DonateBean.java.


2. Ensure that the test environment is running:

- If the Derby Network Server is not running, open a command window, change directory to

-  C:\IBM\SDP\runtimes\base\_v7\derby\bin\networkServer
-  /opt/IBM/SDP/runtimes/base\_v7/derby/bin/networkServer

and execute:

-  startNetworkServer.bat
-  startNetworkServers.sh

- If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.

- If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.

3. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).

## 15.4 Test the messaging artifacts as-is

In this section you test the current messaging artifacts to verify that the DonateMDB onMessage method fails to process the inbound message with Java EE application security enabled:

1. Execute the DonateMDBClient as described in 13.7, “Test the messaging client” on page 514.
2. The DonateMDBClient command output shows that a message is sent:  
  
DonateMDBClient: Establishing connection  
DonateMDBClient: Establishing session  
DonateMDBClient: Sending message  
DonateMDBClient: Message sent!
3. In the workbench Console view, verify that the DonateBean session EJB failed to process the message because the user context was UNAUTHENTICATED:

```
DonateMDB: error in handleMessage: SECJ0053E: Authorization failed for ???  
while invoking (Bean)DonateEAR#DonateEJB.jar#DonateBean  
donateToFund:int,java.lang.String,int:1 null
```

## 15.5 Update the DonateMDBClient and DonateMDB

In this section, we update the DonateMDBClient to put the user ID and password in the JMS header. We also update the DonateMDB onMessage method to extract these values, and then execute the donation as that user.

### 15.5.1 Update the DonateMDBClient code

Here we update the DonateMDBClient to put the user ID and password in the JMS header:

1. In the Enterprise Explorer, open **Main(.java)** (in DonateMDBClient/appClientModule /(default package)).
2. Before sending the message using destSender.send, add two statements to set properties in the JMS header that specify the user ID (DNARMSTRONG or DCBROWN) and password on whose behalf the MDB should execute the DonateBean session EJB:

```
TextMessage toutMessage = session.createTextMessage(args[0]);  
  
/* Added for MDB Security  
toutMessage.setStringProperty("userid", "DNARMSTRONG");  
toutMessage.setStringProperty("password", "xxxxxxx");
```

```
destSender.send(toutMessage);
```

3. Replace “xxxxxxx” with the actual password that you used for DNARMSTRONG in 10.3, “Preliminary security setup” on page 365.
4. Save and close Main.java.

## 15.5.2 Update the DonateMDB code

Next we update the DonateMDB onMessage method to extract these values, and then execute the donation as that user:

1. In the Enterprise Explorer, open **DonateMDB.java** (in DonateEJB/ejbModule/donate.mdb).
2. Locate the onMessage method, and replace the handleMessage(arg0) call with the **F15-1 MDB Security Message** snippet (Example 15-1).

*Example 15-1 F15-1 MDB Security Message snippet and surrounding code*

---

```
public void onMessage(Message message) {
    //handleMessage(message);
    try {
        String tuserid = message.getStringProperty("userid");
        String tpassword = message.getStringProperty("password");
        LoginContext lc = new LoginContext("WSLogin",
            new WSCallbackHandlerImpl(tuserid, null, tpassword));
        lc.login();
        System.out.println("DonateMDB: Subject set to "
            + lc.getSubject().toString());

        // Invoke a Java EE resource using the authenticated subject
        final Message message2 = message;
        WSSubject.doAs(lc.getSubject(), new java.security.PrivilegedAction()
        {
            public Object run() {
                handleMessage(message2);
                return null;
            }
        });

    } catch (Exception e) {
        System.out.println("DonateMDB: Exception: "
            + e.getLocalizedMessage());
        e.printStackTrace();
    }
}
```

---

3. Organize imports (**Ctrl+Shift+O**) to resolve to resolve:



```
import javax.security.auth.login.LoginContext;
import com.ibm.websphere.security.auth.WSSubject;
import com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl;
```

4. Save and close DonateMDB.

#### Behind the scenes:

The processing of the updated onMessage method is as follows:

- Extract the user ID and password from the JMS properties:

```
String tuserid = msg.getStringProperty("userid");
String tpassword = msg.getStringProperty("password");
```

- Create the login context for this user and login:

```
LoginContext lc = new LoginContext("WSLogin",
    new WSCallbackHandlerImpl(tuserid, null, tpassword));
lc.login();
```

- Invoke handleMessage using the authenticated subject, from the login context:

```
WSSubject.doAs(lc.getSubject(),
    new java.security.PrivilegedAction() {
        public Object run() {
            handleMessage(message2);
            return null;
        }
    });
```

WSSubject is an IBM extension to the standard Java EE Subject implementation that is provided to ensure that the correct subject is used for the doAs invocation. Further information can be found in the WebSphere Application Server Information Center at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/rsec\\_jaasauthor.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/rsec_jaasauthor.html)

## 15.6 Test the updated messaging artifacts

In this section, we test the updated messaging artifacts to verify that the MDB `onMessage` method successfully processes the inbound message and passes the security principal to the session bean:

1. Publish the `DonateEAR` enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Execute the `DonateMDBClient` as described in 13.7, “Test the messaging client” on page 514.
3. The `DonateMDBClient` command output shows that a message is sent:

```
DonateMDBClient: Establishing connection
DonateMDBClient: Establishing session
DonateMDBClient: Sending message
DonateMDBClient: Message sent!
```

4. In the workbench Console view, verify that the `DonateMDB` received the message, set the user context to `DNARMSTRONG` (or `DCBROWN`), and successfully invoked the `DonateBean` session EJB:

```
DonateMDB: Subject set to Subject:
Principal: defaultWIMFileBasedRealm/DNARMSTRONG
Public Credential: com.ibm.ws.security.auth.WSCredentialImpl@4f664f66
Private Credential:
com.ibm.ws.security.token.SingleSignonTokenImpl@552a552a
Private Credential:
com.ibm.ws.security.token.AuthenticationTokenImpl@53ce53ce
Private Credential:
com.ibm.ws.security.token.AuthorizationTokenImpl@54d054d0

DonateMDB: Input Message = 1 DonationFund 1
EmployeeFacade: User id=DNARMSTRONG
EmployeeFacade: isCallerInRole=true
Message sent to topic://1 : Request = 1 DonationFund 1 , result = Transfer
successful
DonateMDB: The Result Message is: Transfer successful
```

**Extra Credit:** alter DonateMDB to send other userid and password combinations using tuserid and tpassword:

► **DGADAMS with a valid password:**

```
DonateMDB: Subject set to Subject:
  Principal: defaultWIMFileBasedRealm/DGADAMS
  Public Credential:
com.ibm.ws.security.auth.WSCredentialImpl@3f133f13
  Private Credential:
com.ibm.ws.security.token.SingleSignonTokenImpl@46914691
  Private Credential:
com.ibm.ws.security.token.AuthenticationTokenImpl@42b942b9
  Private Credential:
com.ibm.ws.security.token.AuthorizationTokenImpl@46374637

DonateMDB: Input Message = 1 DonationFund 1
[SECJ0053E: Authorization failed for defaultWIMFileBasedRealm/DGADAMS
while invoking (Bean)DonateEAR#DonateEJB.jar#DonateBean
donateToFund:int,java.lang.String,int:1 Subject:
  Principal: defaultWIMFileBasedRealm/DGADAMS
  ...
  ...

is not granted any of the required roles: DMANAGERS_ROLE DUSERS_ROLE
```

► **DCBROWN with an invalid password:**

```
com.ibm.ws.wim.adapter.file.was.FileAdapter login CWWIM4512E The
password match failed.
```

## 15.7 Explore!

There are no unique Explore resources for this chapter.





## Part 5

# Advanced Java EE 5 and Web 2.0

In Part 5, we perform the advanced tasks described in the following chapter:

- Chapter 16, “Develop a Web 2.0 client” on page 547





## Develop a Web 2.0 client

In this chapter, we provide detailed information on how to develop a rich Web 2.0 application with HTML and JavaScript client code and servlet and Web service logic on the server.

The Web 2.0 implementation is using the **WebSphere Application Server Feature Pack for Web 2.0**, which provides Asynchronous JavaScript and XML (Ajax), JavaScript Object Notation (JSON), Representational State Transfer (REST), and the Dojo toolkit.

## 16.1 Learn

In this section we describe the basics of Web 2.0 technologies and how these technologies are implemented in the WebSphere Application Server Feature Pack for Web 2.0.

### 16.1.1 What is a Web 2.0 client?

A Web 2.0 client is a Web application that has enhanced desktop-like responsiveness and a much more intuitive and participatory user interface. The onset of mass availability of high speed Internet access combined with the powerful modern client machines with much more processor power and memory, seems to have created this new client-server paradigm where richer Web 2.0 applications are now replacing the traditional thin client (HTML only) Internet applications. These Web 2.0 applications, when run on modern Web browsers, are now capable of delivering rich user interfaces, which were before only possible with heavyweight thick graphical user interface (GUI) applications that ran on the native operating system.

Many of the popular Web sites are using Web 2.0 technologies to provide users a more desktop-like interface, thus turning the Web browser into a pervasive operating system. The Web 2.0 technologies have been the main reason behind the success of some of these Web portals, as users have embraced them due to their superior user interface features.

In a Web 2.0 application, the presentation handling is delegated to the client, where browsers use widgets developed using Web 2.0 technologies to render the data sent by the server. Therefore, the server is now only responsible for executing the program logic and to prepare and send the data requested by the client.

#### **Business benefits of a Web 2.0 client**

Some of the business benefits of adding Web 2.0 capabilities to Web applications include:

- ▶ A more interactive, differentiated experience that can lead to longer sessions and increased customer loyalty.
- ▶ Traditional Web applications have suffered from poor interactivity and responsiveness towards end users. Responsiveness, local actions which can result in fewer abandoned transactions, higher completion rates, and higher end-user productivity.
- ▶ Interaction in classic Web applications is based on a multi-page interface model, in which for every request the entire page is refreshed. In the Web 2.0



applications, the user interface conforms to the single page model or, in other words, does not follow the concept of multiple Web pages. The page can now instead be called a *view*, which is composed of several individual components that are updated or replaced independently, so that the entire page does not have to be reloaded on each user action. This, in turn, helps to increase the levels of interactivity, responsiveness, and user satisfaction.

### 16.1.2 Technologies for Web 2.0 client development

The Web 2.0 or rich Internet applications are not new to us. Most of the Java developers that embraced the Internet during its infancy period were particularly drawn to it because of the Java applet technology. Java applets were introduced in 1995 as part of the first version of Java by Sun Microsystems. Applets provided the ability to run dynamic data driven software applications on the browser, which was not possible before on a mainly static HTML only Internet. Applets also provided richer user interface previously available only with native applications.

Java applets were software applications or components that were written in Java and ran on the browser using the Java Virtual Machine (JVM). The applet code was hosted on the server but was transferred to the client (browser) in a bytecode when the applet was first executed. Applets ran completely on the client and interacted with the Java based applications on the server to get their data.

The new Web 2.0 or rich Internet applications follow a very similar client-server paradigm. Applets also enjoyed instant stardom, just like Web 2.0 technologies, and thousands of applet based applications were developed. These applications ranged anywhere from small utilitarian applications such as mortgage calculators to some big applications such as bank teller and call center applications.

Applet based dynamic applications finally gave way to thin-client architecture where application servers executed most of the logic on the server side and generated HTML which was then sent to the browser. With some research you can find several reasons cited for failure of Java applets to become successful on the client, but we think some of the important reasons were that networks were much slower, the browser and JVM were in their first few generations, and the client computers were not nearly as powerful as they are today.

Today's personal computers are much more powerful than some servers used a decade ago. The browsers are now in their eighth generation and runtime environments available on the client are now also much more powerful.

Following are some of the main client side technologies that can be used to develop richer user interface for Web applications:

- ▶ Asynchronous JavaScript and XML (Ajax)
- ▶ Flex with Adobe® Flash browser plug-in

- ▶ JavaFX by Sun Microsystems
- ▶ Silverlight by Microsoft
- ▶ Rich client platform (RCP) by Eclipse Foundation
- ▶ XUL by Mozilla Foundation

## Asynchronous JavaScript and XML (Ajax)

The term *Ajax* is a short form of Asynchronous JavaScript and XML. Ajax is an asynchronous programming pattern that is primarily based on JavaScript and XML technologies. Ajax is an open technique for creating rich user experiences in Web-based applications that does not require additional browser plug-ins. Ajax interfaces result in increased customer satisfaction and line of business productivity by streamlining end-user interaction with Web-based applications.

The other technologies in Ajax include XHTML, DHTML, and CSS, which provide a standards-based presentation. JavaScript is the programming language that is used to bind everything together. The Document Object Model (DOM) provides dynamic display and interaction. And XML can be used as the format for data transfer between client and server, although any text based pre-defined format can be used such as HTML, plaintext, and JavaScript Object Notation (JSON).

The core of Ajax technology is the JavaScript object `XMLHttpRequest`. With an `XMLHttpRequest` object, you can use JavaScript to make a request to the server and process the response. It is the JavaScript technology through the `XMLHttpRequest` object that talks to the server, which is not the normal application flow and this is where Ajax gets much of its magic.

Figure 16-1 on page 551 illustrates an asynchronous request invocation with Ajax. In a traditional thin client Web application, when you fill out a form and submit it, the entire form is sent to the server. The server then processes the request (using server side technologies, such as PHP, CGI, Java EE, or .NET, and sends back the markup of a new Web page. With Ajax, JavaScript technology is used to intercept the form data and send that data to a JavaScript code. JavaScript code then uses the `XMLHttpRequest` object to talk to the server side code. The server then sends data back to a callback JavaScript method that decides what to do with that data and how to render it on the client.

This object can use DOM to manipulate the structure of the existing HTML document on the browser and thus, update data elements, such as `<div>`, or update form fields on the fly, giving that desktop-like responsiveness and intuitive interaction to the application. Ajax updates the portion of the HTML page without refreshing the whole HTML page, and all without user intervention! This is the power of `XMLHttpRequest`, which can talk seamlessly back and forth with a server. The result is a dynamic, responsive, highly-interactive experience such as a desktop application, but with all the power of the Internet behind it (Figure 16-1).

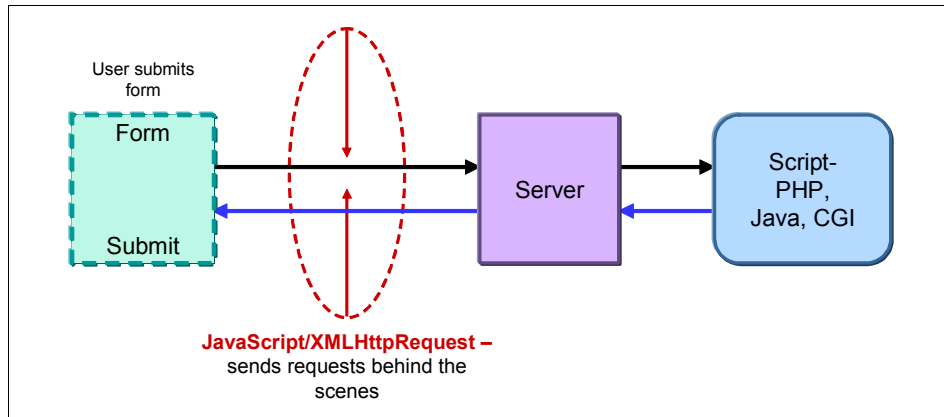


Figure 16-1 Client-server communication using the XMLHttpRequest object

Thus, Ajax bridges the gap between desktop applications and Web applications. Web pages are more responsive by seamlessly exchanging small amounts of data with the server. The entire Web page does not have to be reloaded each time a change is made, thus increasing Web page interactivity, speed, and usability. Ajax applications leverage standard browser features and do not require the installation of browser plug-ins.

Ajax can build clients that can front any server side application, such as Java EE, .NET, PHP, CGI, and so forth.

## JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a lightweight data interchange format. Just like XML, it is text based and easily readable by humans and can be easily parsed by machines. The JSON format was specified by Douglas Crockford in RFC-4627.

Markup languages such as XML and HTML were initially built to give semantic meaning to text with in documents. Example 16-1 shows how markup can be used to provide specific meaning to a piece of a document:

Example 16-1 XML snippet

```

<From>Maan Mehta</From>
<To>Ueli Wahli</To>
<Subject>This is a test message</Subject>
<Body>This is the body of the message. This is an
<b>important</b> message.</Body>
  
```

XML and especially HTML have done good job for this purpose. Over the last ten years, XML has also become a defacto standard markup language for message transfer between different systems and has played a very vital role in the evolution of Web services. On the other hand, HTML and its variants, such as XHTML and WML, have become a defacto markup languages for sending the data from the applications running on servers to browser-based clients.

JSON is a subset of JavaScript and is based on its array, string, and object literal syntax. JSON's format is language independent and its syntax is familiar to programmers of the C-family of languages, such as C, C++, Java, JavaScript, and Python. This has made it a good choice and an alternative to XML to describe data structures. Therefore, JSON can be used to serialize a data structure and transfer it over a network connection to the client (browser), where an Ajax application can deserialize the content into a JavaScript object by using a simple `eval` statement.

The following code snippet is the JSON representation of a JavaScript object that encapsulates the details of a person. The object has string attributes or fields for name, address, phone, and e-mail:

```
{
  "name": "Maan Mehta",
  "address": {
    "street": "27 Some Street",
    "city": "Toronto",
    "province": "ON",
    "postalCode": "M42 2M2"
  },
  "phone": "416-123-4567",
  "email": "maan@somewhere.ca",
}
```

If this string is contained in a JavaScript string variable called `person`, then we can create the JavaScript object by using the following `eval` statement:

```
var someOne = eval('(' + person + ')');
```

We can then access the value of the fields such as name, city, and email using the JavaScript dotted notation as follows:

```
var name = someOne.name;
var city = someOne.address.city;
var email = someOne.email;
```

On the server side, several parsers are available to serialize server side objects into the JSON format. IBM's Web 2.0 feature pack for WebSphere provides the JSON4J library to serialize the contents of the Java objects into the JSON format and vice-a-versa.

## Representational State Transfer (REST)

Representational State Transfer (REST) is an architectural style for distributed systems. REST design the systems by defining its resources, these resources have a simple standard interface to be manipulated and to exchange its representations.

When applied in the Web, REST identifies resources by URIs, and it relies on the HTTP protocol for standard actions to manipulate the representations of the URIs. For example, a Web page is a resource that has a URI to be accessible. The HTTP method GET retrieves the representation of that Web page to a client.

The HTTP methods are mapped to actions on the resources as shown in Table 16-1.

*Table 16-1 HTTP methods and concepts*

Action	HTTP
Create	PUT
Read	GET
Update	POST
Delete	DELETE

### 16.1.3 WebSphere Feature Pack for Web 2.0

The IBM WebSphere Application Server Feature Pack for Web 2.0 extends the reach of service-oriented architecture (SOA) by connecting external Web services, internal SOA services, and Java EE objects into highly interactive Web application interfaces. To reduce IT costs and speed time to market, it provides a supported, open Ajax development framework that leverages existing SOA and Java EE assets to deliver rich user experiences.

Highlights and capabilities of the Feature Pack for Web 2.0 include:

- **Web 2.0 to SOA connectivity:** For enabling connectivity from Ajax clients and mash-ups to external Web services, internal SOA services, and Java EE assets. It also extends enterprise data to customers and partners through Atom Syndication Format (Atom) and Really Simple Syndication (RSS) Web feeds.

- ▶ **Ajax messaging:** For connecting Ajax clients to real-time updated data such as stock quotes or instant messaging.
- ▶ **Ajax development toolkit:** The Ajax development toolkit for WebSphere Application Server based on the Dojo toolkit with IBM extensions:
  - An enterprise-ready Ajax toolkit for building richer, more interactive Web sites (through the Dojo toolkit and IBM extensions).
  - Support for end-to-end Ajax patterns, including Java EE server integration (REST Web remoting, proxy services, Ajax messaging).
  - Lightweight data interchange in a format that is more easily consumable by Ajax clients (JSON4J library).
  - The ability to asynchronously push events through the use of the RSS and Atom.
  - Samples and tutorials for building end-to-end Ajax applications, using WebSphere Application Server products.
  - Support for tagging and feeds using RSS and Atom within Web applications.

The Feature Pack for Web 2.0 includes the following functionality:

- ▶ **Ajax client runtime:** Static content that can be added to either a WAR file or Web server.
- ▶ **Dojo toolkit:** JavaScript toolkit for creating Ajax client-side applications:
  - <http://dojotoolkit.org/>
- ▶ IBM extensions to the Dojo toolkit that include additional widgets and client libraries:
  - **IBM Atom library:** This library provides client-side support of Atom feeds in the browser and allows for two-way communication with those feeds using the Atom Publishing Protocol (APP). Besides the base library, this also includes a reference implementation application of the dojo.data APIs and a collection of three Atom widgets for a rich viewing and editing of Atom feeds.
  - **IBM Gauge Widget library:** This library includes a pair of widgets (AnalogGauge and BarGraph) for displaying numerical data using Scalable Vector Graphics (SVG) or Vector Markup Language (VML), depending on the browser.
  - **IBM SOAP library:** This library is a Dojo toolkit extension that makes it easy for an application to connect to an existing SOAP-based Web service. Using this library you can make remote procedure calls to invoke the Web service methods directly from the client, the browser.

- **IBM Open Search library:** This library makes it easy to invoke any Open Search-compliant service and to bind search results to widgets within the Ajax application
- ▶ **Ajax connectivity:**
  - Ajax proxy: This proxy can be used to broker client requests from multiple Domains while using Ajax. JavaScript sandboxing rules prevent the execution of network requests to servers from where the JavaScript did not originate. As an example, if the JavaScript application originated from domain A and attempts to use an XMLHttpRequest to domain B, the browser will prevent the domain B request. The proxy can be used to broker the request, which provides the appearance to the client that the request came from the same server from which the JavaScript originated.
  - Remote procedure call (RPC) adapter.
  - Abdera-based library support.
- ▶ **Ajax developer's guide**
- ▶ **JSON4J library**
- ▶ **Asynchronous Web 2.0 messaging service:** The Web messaging service is a publish or subscribe implementation connecting the browser to the WebSphere Application Server service integration bus (SIB) for server-side event PUSH to the browser. The Web messaging service can enable a Web service or any other item connected to the SIB to PUSH data to the browser. You can use the Web messaging service in a new or existing application by placing a utility file library (JAR) in an application Web module, setting up a simple configuration file, and configuring servlet mappings.

## JSON4J library

JSON4J library is an implementation of a set of JSON handling classes for use within Java environments. The JSON4J library provides the following functions:

- ▶ A simple Java model for constructing and manipulating data to be rendered as the JSON implementation.
- ▶ A fast transform for XML to JSON conversion, for situations where conversion from an XML reply from a Web service into a JSON structure is desired for easy use in Ajax applications. The advantage of this is that Ajax-patterned applications can handle JSON formatted data without having to rely on ActiveX objects in Microsoft Internet Explorer XML transformations and other platform-specific XML parsers. In addition, JSON-formatted data tends to be more compact and efficient to transfer.

- A JSON string and stream parser that can generate the corresponding JSONObject, which represents that JSON structure in Java. You can then make changes to that JSONObject and serialize the changes back to the JSON implementation.

The API is distributed as an optional library and set of documentation that is included in the Feature Pack for Web 2.0. The library includes the Java class files packaged in a Java archive (JAR) format, JSON4J.jar, in the lib folder of the stand-alone package of the feature pack. This library can be used for stand-alone Java applications or Java EE Web applications that have to parse and handle JSON text. For Java EE Web applications, package the JSON4J.jar file in the WEB-INF/lib directory of the Web application archive (WAR) file.

The JSON4J library provides the following two important classes:

```
com.ibm.json.java.JSONArray  
com.ibm.json.java.JSONObject
```

JSONObject class has a static method called parse that takes a parameter of type String and if that string is of valid JSON format, it creates and returns a deserialized Java Object. The JSONObject class also provides get and put methods to retrieve or set the values of properties of the Java object of type JSONObject. Example 16-2 illustrates the usage of these methods.

---

*Example 16-2 Using parse and get method of the JSONObject Class*

---

```
String jsonString = "{\"name\":\"Maan Mehta\", \"phone\":\"416-123-4567\"}";  
JSONObject javaObject = JSONObject.parse(jsonString);  
String fullName = javaObject.get("name");  
String phoneNumber = javaObject.get("phone");  
System.out.println("Name is: " + fullName + " & the Phone is: " + phoneNumber);
```

---

The output of code snippet in Example 16-2 is:

```
Name is: Maan Mehta and the Phone is: 416-123-4567
```

The JSONObject class also provides a serialize method that serializes and returns the Java object as a string in the JSON format. Example 16-3 illustrates the usage of the serialize method.

---

*Example 16-3 Using the serialize method of the JSONObject class*

---

```
String jsonStr = javaObject.serialize(true);  
System.out.println("JSON representation of the Java Object is:\n" + jsonStr);
```

---



The output of code snippet in Example 16-3 on page 556 is:

```
JSON representation of the Java Object is:
{
  "name": "Maan Mehta",
  "phone": "416-123-4567"
}
```

## **Feature Pack sample applications**

The Feature Pack also includes the following sample applications that come with complete source code and ready to deploy and run on WebSphere Application Server or WebSphere Application Server Community Edition.

### ***Feed samples***

Atom Syndication Format and RSS are two standardized specifications used to deliver content feeds. Apache Abdera is the open-source project has been chosen to provide feed support within WebSphere Application Server.

Apache Abdera addresses and provides support for:

- ▶ Reading RSS content
- ▶ Atom syndication format
- ▶ Atom publishing protocol

The feed samples demonstrate these capabilities and have simple examples to read an RSS feed, create an atom feed, and read atom content.

### ***Courier application sample***

This sample Web application include simple examples that domesticate the following capabilities of RPC adapter:

- ▶ Handling of JSON response using JSON service
- ▶ Handling of XML response using XML RPC
- ▶ White and black list methods, and
- ▶ Specifying validators

The courier application is also available as an Eclipse plug-in that can be imported into an Eclipse workspace.

### ***Web 2.0 Plants by WebSphere sample***

The Feature Pack comes with a modified version of the well known Plants by WebSphere sample Java EE Web application from IBM. The Plants by WebSphere application is a fictional Web site that makes available nursery items such as vegetables, flowers, accessories, and trees. You can view an online catalog, select items, and add them to a cart. When the cart contains items, you can proceed to log in, supply credit card information, and check out.

In this sample, Dojo widgets have been used to enhance the user interface of the application. A new Dojo widget has also been created to enable drag and drop capabilities for the shopping cart. The `dojo.xhr` (GET/POST) calls have been used for the client-server communication and on the server side, an RPC adapter layer has been created to map traditional Java EE constructs, such as Enterprise Java Beans (EJBs), Web services, and POJOs, to lightweight constructs, such as JSON or XML using JSON4J and XML RPC.

### ***Quote Streamer sample***

This sample application demonstrates usage of the Web 2.0 messaging service by pushing realtime updates of stock quotes to the browser using the cometd transport capabilities.

## **16.2 Web 2.0 scenario use cases**

In this section, we define the scenario use cases for the Donate sample application based on Web 2.0 technologies.

### **16.2.1 Use case requirements**

We follow the same use cases that we have used in other chapters. The objective of the exercises in this chapter is to give you a head start in developing a Web 2.0 client for a Java EE server application using the Feature Pack for Web 2.0.

We developed two use case scenarios: Find Employee and Donate Vacation.

#### **Find Employee**

In this use case scenario (Figure 16-2), the user selects an employee from the list of Employees pre-populated in a drop-down and clicks **Find** to retrieve and view the employee details. The employee details consist of first and last name and the number of vacation hours remaining.

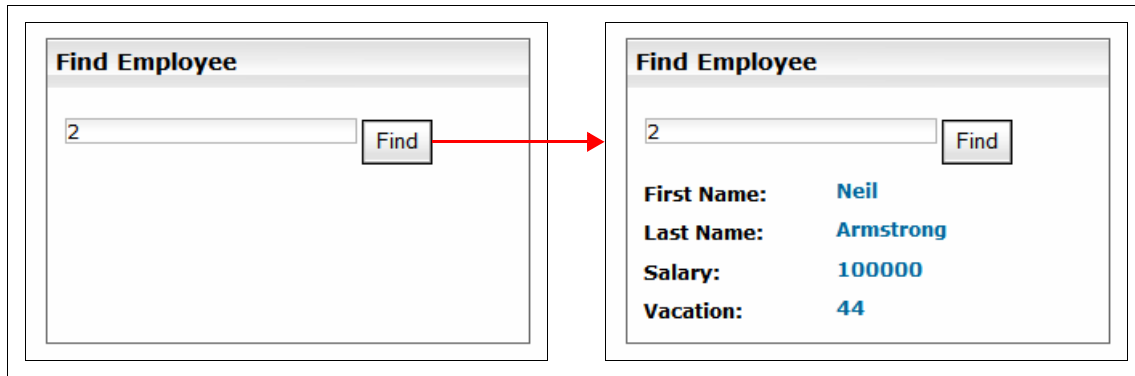


Figure 16-2 Find Employee use case

## Donate Vacation

In this use case scenario, the employee selects the fund to donate vacation hours, enter the number of hours to donate, and click **Donate**.

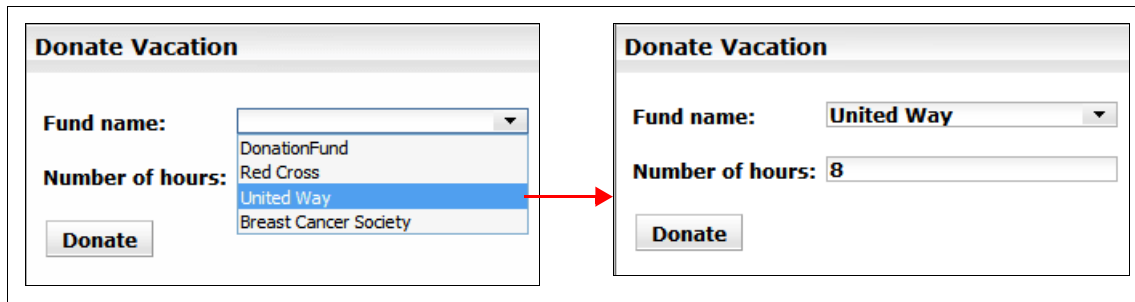


Figure 16-3 Donate Vacation use case

If the transaction is successful, a success status message is displayed in a modal dialog (Figure 16-4). The status message also displays the remaining vacations hours. If there is an error during the transaction, the modal dialog displays the error message instead.

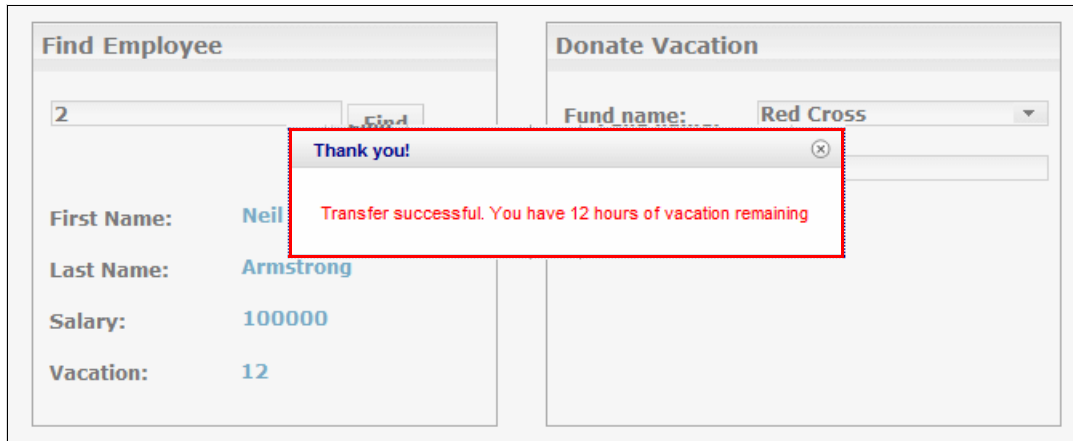


Figure 16-4 An Ajax modal dialog displays the status message

## 16.2.2 Rich user interface requirements

This section highlights the rich user interface requirements that we are going to deliver using the features provided by the Feature Pack for Web 2.0.

- ▶ Single page model: No page flicker and reloading of the whole page.
- ▶ Pre-population of drop-down fields: To eliminate errors, we pre-populate the donation fund names field (Figure 16-3).
- ▶ Type-ahead support: The input field for the donation fund name supports the type-ahead feature (Figure 16-5 on page 560).

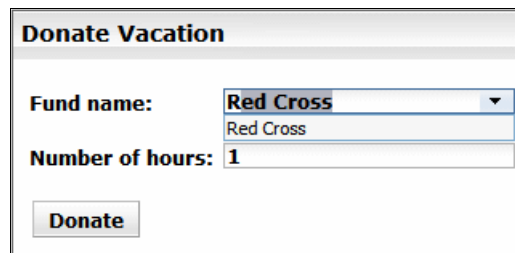


Figure 16-5 Type-ahead support to select the fund name

- ▶ Client-side validation: We use out-of-the-box input validation features provided by Dojo form widgets and also implement custom validation logic. For example, the employee number has a range constraint, and the fund name must be from those listed in the pre-populated drop-down field (Figure 16-6), and the amount of vacation an employee can donate should be less than their remaining vacation.

Find Employee		Donate Vacation	
<input type="text" value="9"/> <span>This value is out of range.</span>		Fund name: <input type="text"/>	
First Name: <b>Neil</b> Last Name: <b>Armstrong</b> Salary: <b>100000</b> Vacation: <b>12</b>		Number of hours: <input type="text" value="1"/> <input type="button" value="Donate"/>	

Find Employee		Donate Vacation	
<input type="text" value="2"/> <input type="button" value="Find"/>		Please select a valid donation Fund <input type="text" value="XYZ"/>	
First Name: <b>Neil</b> Last Name: <b>Armstrong</b> Salary: <b>100000</b> Vacation: <b>12</b>		Number of hours: <input type="text" value="8"/> <input type="button" value="Donate"/>	

Figure 16-6 Input field validation

- Ajax modal dialogs for status and error messages (see Figure 16-4 on page 560).

## 16.3 Installing the Feature Pack for Web 2.0 v1.0.0.1

### Behind the scenes:

This feature pack will already be installed if you installed the Rational Application Developer trial code as described in Chapter 2, “Install and configure software” on page 25. In this case, you can skip ahead to 16.4, “Jump start” on page 564.

The Feature Pack for Web 2.0 v1.0.0.1 is an optionally installable product extension for IBM WebSphere Application Server. The Feature Pack for Web 2.0 is also available for IBM WebSphere Application Server v6.0, v6.1, WebSphere Application Server Community Edition v2.0 and v2.1 in addition to IBM WebSphere Application Server v7.0.

This feature pack will already be installed if you installed the Rational Application Developer trial code as described in Chapter 2, “Install and configure software” on page 25. In this case, you can skip ahead to 16.4, “Jump start” on page 564.

- You can download the Feature Pack by visiting the following URL for the IBM Product page for this feature pack:

<http://www.ibm.com/software/webserver/appserv/was/featurepacks/web20/features/>

- On this page, click **Download Feature Pack for Web 2.0 now**. You are asked to authenticate using IBM credentials. After authentication, you are taken to the download page where you are presented with options to download the feature pack relevant to your application server and its version. Select the option to download for **Feature Pack for Web 2.0 Version 1.0 Fix Pack 2 for WebSphere Application Server Version 1.0.0.2** and click **Continue** (Figure 16-7).

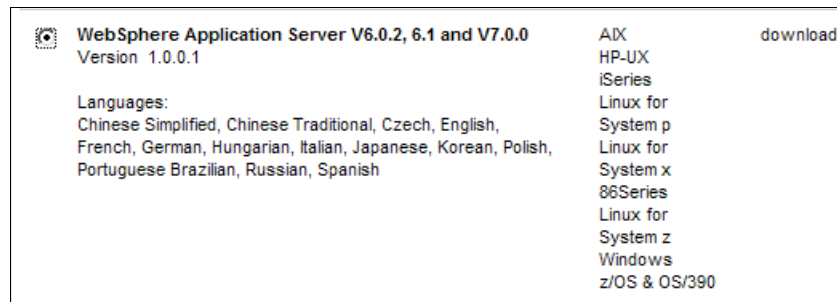


Figure 16-7 Select to download for the appropriate application server

- On the following Web page, select to download the feature pack:
  - 1.0.0-WS-WASWeb20-FP0000002.pak (226 MB)
  - Click **Download now**.
- The feature pack is installed using the WebSphere Application Server Update Installer and the most recent version must be downloaded installed from this web site:

<http://www.ibm.com/support/docview.wss?uid=swg24020448>

## Installation of the Feature pack for Web 2.0 on Windows

The installation of the Feature Pack for Web 2.0 Version 1.0 is the same for WebSphere Application Server, Versions 6.0.2.X, 6.1.0.X, and 7.0.0.X. The Fix Pack is built as an Update Installer for WebSphere Application Server maintenance package and is installed with that program in the same manner as other Fix Packs for WebSphere Application Server.

The overall process of installing a feature pack (or in general any fix pack) is described at:

- WebSphere Application Server Information Center: Installing fix packs using the graphical user interface

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.installation.nd.doc/info/ae/ae/tins\\_updi\\_fixpk.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.installation.nd.doc/info/ae/ae/tins_updi_fixpk.html)

Table 16-2 describes some of the folders created under the install root of the Feature Pack.

*Table 16-2 Feature Pack for Web 2.0 contents*

Directory	Description
ajax-rt_1.X	Ajax client runtime contains a development copy of the Dojo toolkit plus IBM extensions. Copy these into your application WAR files as static resources, or place them in your Web server as static resources and reference them from your application.
samples\AjaxHello	Sample application to illustrate the basics of developing applications with the Dojo toolkit.
documentation\DeveloperGuide	A starting point for learning about Ajax Development, debugging, and usage patterns.
optionalLibraries\Feed	Abdera libraries for ATOM document processing.
samples\Feed	Samples created by IBM to illustrate examples of feed usage.
optionalLibraries\JSON4J	JSON4J library, set of utility classes intended for use in a variety of Java applications that serialize and deserialize JavaScript Object Notation (JSON) formatted data.
optionalLibraries\MessagingService	Client library for using the Web messaging feature.
samples\PlantsByWebSphere	Sample application that provides the Web 2.0 client for the WebSphere's sample application.

Directory	Description
samples\QuoteStreamer	Sample application that demonstrates the usage of the Web messaging service by displaying realtime updates of stock quotes using the cometd transport capabilities.
optionalLibraries\RPCAdapter	IBM RPC Adapter library for Web-remoting.
samples\RPCAdapter	Sample applications for the RPCAdapter feature, CourierApp and HelloWorld.
samples\SoapSample	Sample application to illustrate the basics of invoking a SOAP service using the IBM SOAP extension of the Dojo toolkit.

## 16.4 Jump start

If you want to start with this chapter and skip some or all of the preceding chapters, follow the instructions in Appendix A, “Jump start” on page 615 to create the starting configuration for this chapter.

For this chapter to work, make sure that you have completed all the applications of Part 2, “Core Java EE application” on page 121.

You can also start with the finished applications of Part 3, “Web services” on page 407 or Part 4, “Messaging” on page 475.

## 16.5 Disable Java EE EJB application security

This chapter does not contain the application code required to support a secure session EJB. Therefore, before starting this chapter, we have to disable the Java EE EJB security:

1. Disable declarative security in the DonateBean session EJB:
  - a. In the Enterprise Explorer, open **DonateBean** (in DonateEJB/ejbModule/donate.ejb.impl).
  - b. Before the two donateToFund methods, comment out the @RolesAllowed statement and add a @PermitAll statement.

```
//@RolesAllowed( { "DMANAGERS_ROLE", "DUSERS_ROLE" })
@PermitAll
public String donateToFund(Employee employee, .....
```








```

.....

//@RolesAllowed( { "DMANAGERS_ROLE","DUSERS_ROLE" })
@PermitAll
public String donateToFund(int employeeId, .....)
```

- c. Save and close DonateBean.java.
2. Disable programmatic security in the EmployeeFacade session bean:
  - a. In the Enterprise Explorer, open **EmployeeFacade** (in DonateEJB/ejbModule /vacation.ejb.impl).
  - b. Comment the two statements:
 

```

//if (!ctx.isCallerInRole("DMANAGERS_ROLE"))
// clone.setSalary(null);
```
  - c. Save and close EmployeeFacade.java.
3. Ensure that the test environment is running:
  - If the Derby Network Server is not running, open a command window, change directory to:
    -  C:\IBM\SDP\runtimes\base\_v7\derby\bin\networkServer
    -  /opt/IBM/SDP/runtimes/base\_v7/derby/bin/networkServer
  - and execute:
    -  startNetworkServer.bat
    -  startNetworkServers.sh
  - If the **ExperienceJEE Test Server** is not running, select the server in the Servers view and click the **Start** icon  or select the server, right-click and select **Start**.
    - If the server instance starts successfully, you will find the message WSVR0001I: Server server1 open for e-business in the Console view.
4. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).

## 16.6 Setup the development environment

In this section we prepare a Web project for the development of the Web 2.0 sample application.

## 16.6.1 Create a Dynamic Web project

We use a new Dynamic Web project for the Web 2.0 sample application:

1. In the workbench action bar, select **File** → **New** → **Dynamic Web Project**.
2. At the New Dynamic Web Project pop-up:
  - a. Type **DonateWeb20** as the Project name.
  - b. Select **2.5** for Dynamic Web Module version.
  - c. Select **Default Configuration for IBM WebSphere Application Server v7.0** for Configuration.
  - d. Select **Add project to an EAR** and select **DonateEAR** as the EAR Project Name.
  - e. Click **Next**.
3. At the Web Module pop-up, leave the default values, select **Generate deployment descriptor**, and click **Finish**.
4. At the Open Associated perspective pop-up, click **Yes** to switch to the Web perspective.

## 16.6.2 Enable the web 2.0 facets

Web 2.0 enabled Web projects hold all of the Web resources that are created and used when developing a Web 2.0 Web application.

1. In the Enterprise Explorer, select **DonateWeb20**, right-click and select **Properties**.
2. At the Properties pop-up, on the left select **Project Facets**.
3. At the Properties pop-up, on the right under Project Facets enable the **Web 2.0** facet. (The Ajax Proxy, Dojo toolkit and Server side technologies sub options will be automatically enabled as well). Do **NOT** click OK or Apply at this point.

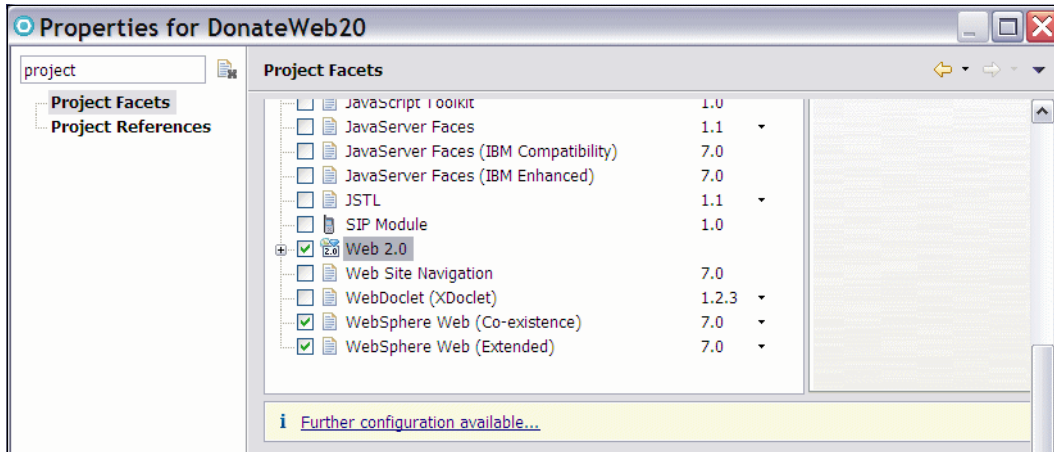


Figure 16-8 Enable the Web 2.0 facet

4. Click the **Further configuration available...** link at the bottom of the Properties pop-up.
  - a. At the Modify Faceted Project pop-ups, view the Web 2.0 facet configuration information and click **OK** to close the pop-up.

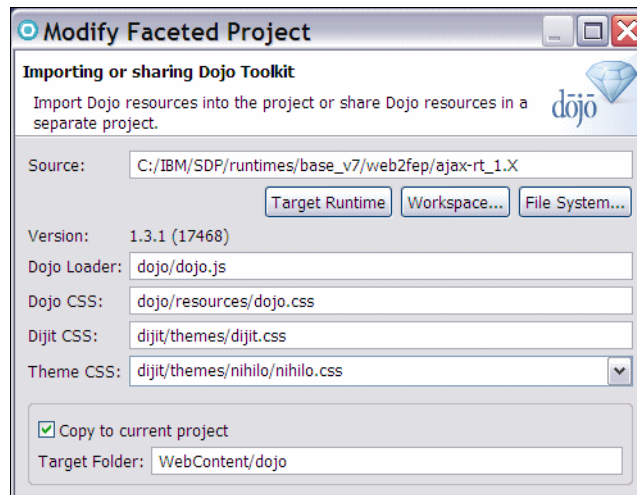
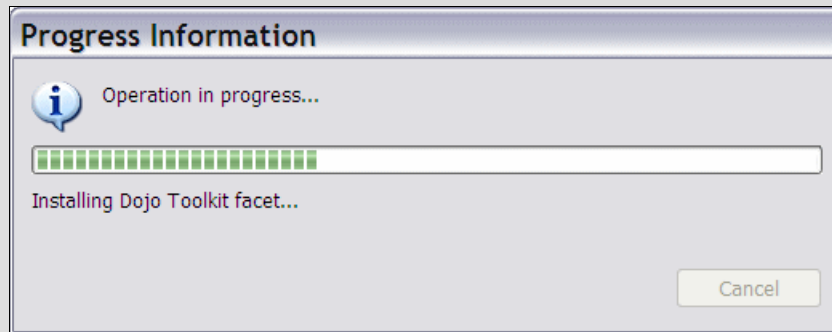


Figure 16-9 Web 2.0 facet configuration

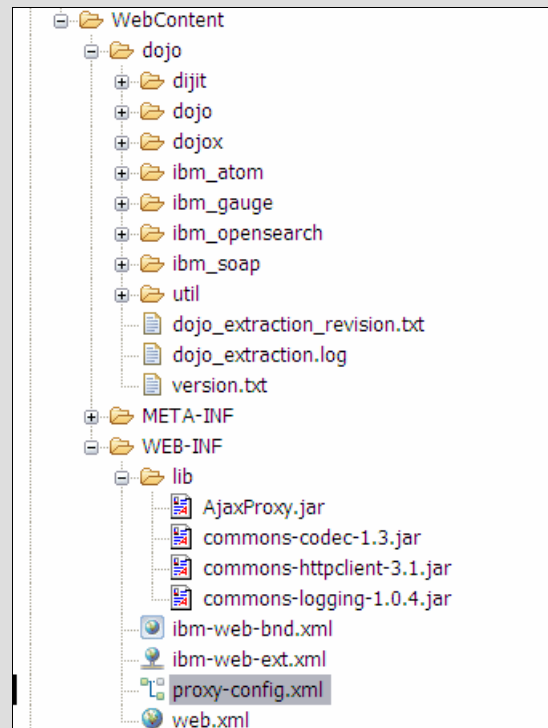
5. Back at the Properties pop-up, click **OK** to save the changes and close the properties dialog.

### Behind the scenes:

The workbench will take a few minutes to install the Web 2.0 artifacts into your project:



The artifacts are copied into the project WebContent directory.



### 16.6.3 Add the dependency to the EJB project

In this Web project, we access an EJB session bean. Therefore, we have to set up the dependency of the Web project to the EJB project.

By adding the DonateWeb20 project to the DonateEAR enterprise application, a dependency has been created in DonateEAR. Now we can set up further dependencies between projects contained in DonateEAR.

1. Select **DonateWeb20**, right-click and select **Properties**.
2. At the Properties pop-up:
  - a. On the left, select **Java EE Module Dependencies**
  - b. On the right, enable the **DonateEJB.jar** selection and click **OK**.

### 16.6.4 Populate the Fund table

To demonstrate some of the features of Web 2.0 components, we have to populate the FUND table of the Donate database with additional fund names:

1. Switch to the **Database Development** perspective.
2. In the Data Source Explorer, select **Database Connections** → **Donate ()**, right-click and select **Open SQL Scrapbook**.

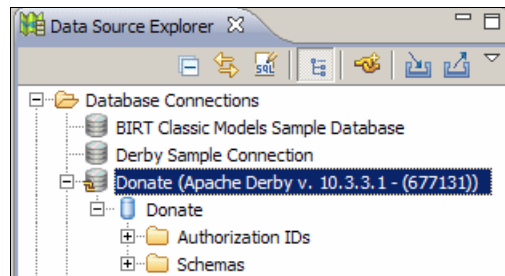


Figure 16-10 Donate database connection

**Off course?** if the Donate connection is not connected, select it, right-click and select **Connect**.

3. At the resulting SQL Scrapbook 2 editor:
  - a. Insert the **F16.1 Web 2.0 Populate Fund Table** snippet into the editor:

```
INSERT INTO "DONATE"."FUND" ("NAME", "BALANCE") VALUES ('Red Cross', 1000);
```

```
INSERT INTO "DONATE"."FUND" ("NAME", "BALANCE") VALUES ('United Way',
1000);
INSERT INTO "DONATE"."FUND" ("NAME", "BALANCE") VALUES ('Breast Cancer
Society', 1000);
```

**Off course?** if you do not see the Snippets view it can be opened by the following steps:

- ▶ In the workbench action bar, select **Window** → **Show View** → **Other**.
- ▶ At the Show View pop-up, select **General** → **Snippets** (or start typing snippet to find the view).
- ▶ Click **OK** and the Snippets view is visible in the lower right pane.

- b. Select anywhere inside the script, right-click and select **Execute All**.
- c. Switch to the SQL Results view in the bottom right and verify that the insertions completed successfully:

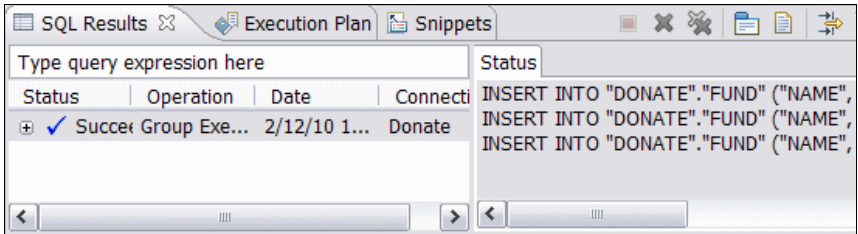


Figure 16-11 SQL insertion results

- d. Close the SQL Scrapbook 0 editor (no save is required).
- 4. Optionally add more employees to the EMPLOYEE table by running the **F16.2 Web 2.0 Populate Employee Table** snippet against the Vacation database.

## 16.7 Implement the Find Employee use case

In this section we implement the Find Employee use case using the REST technology.

### 16.7.1 Create a REST servlet

Here, we create a REST servlet that retrieves the details of an employee by interacting with the EmployeeFacade session EJB through simple EJB injection and access. This servlet then serializes the state of the Employee Java object into

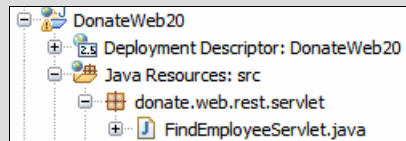
a JSON string and writes that string to the HTTP response output stream. The Web 2.0 client components then call this servlet to retrieve and display the details of the Employee object:

1. Back in the Web perspective, in the Enterprise Explorer, select **DonateWeb20**, right-click and select **New** → **Servlet**.
2. At the Create Servlet pop-up, provide the following data:
  - Java package: **donate.web.rest.servlet**
  - Class name: **FindEmployeeServlet**
  - Superclass: Accept `javax.servlet.http.HttpServlet`
- a. Click **Finish**.

### Behind the scenes:

In creating a servlet, a servlet class and a corresponding mapping in the `web.xml` are created:

- The servlet class is created in the `donate.web.rest.servlet` package.



- The corresponding entries are inserted into the Web deployment descriptor (`DonateWeb20/WebContent/WEB-INF/web.xml`).

```
<servlet>
  <description></description>
  <display-name>FindEmployeeServlet</display-name>
  <servlet-name>FindEmployeeServlet</servlet-name>
  <servlet-class>donate.web.rest.servlet.
    FindEmployeeServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FindEmployeeServlet</servlet-name>
  <url-pattern>/FindEmployeeServlet</url-pattern>
</servlet-mapping>
```

3. In the resulting `FindEmployeeServlet.java` editor, delete all the generated method skeletons (default constructor, `doGet`, and `doPost`), leaving the base class definition and the `serialVersionUID`.

```
public class FindEmployeeServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
```

```
}
```

4. Insert the **F16.3 Web 2.0 Find Employee REST Servlet** snippet before the closing brace (Example 16-4).

**Off course?** if you do not see the Snippets view it can be opened by the following steps:

- ▶ In the workbench action bar, select **Window** → **Show View** → **Other**.
- ▶ At the Show View pop-up, select **General** → **Snippets** (or start typing snippet to find the view).
- ▶ Click **OK** and the Snippets view is visible in the right pane (and you can move it to the lower pane if desired).

5. Organize imports (**Ctrl+Shift+O**)
  - a. At the Organize Imports pop-up, select **com.ibm.json.java.JSONObject** and click **Finish**.

**Behind the scenes:** the following imports are resolved:

```
import java.io.IOException;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import vacation.ejb.interfaces.EmployeeFacadeInterface;
import vacation.entities.Employee;
import com.ibm.json.java.JSONObject;
```

6. Save and close FindEmployeeServlet.java.

*Example 16-4 F16.3 Web 2.0 Find Employee REST Servlet snippet and surrounding code*

---

```
public class FindEmployeeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB
    EmployeeFacadeInterface employeeFacade;

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request,
```



```

        HttpServletResponse response) throws ServletException, IOException {

    String empId = (String) request.getParameter("empId");

    Employee employee = null;
    try {
        employee =
employeeFacade.findEmployeeById(Integer.parseInt(empId));
    } catch (Exception e) {
        e.printStackTrace();
    }

    if (employee == null) {
        response.getWriter().println("Employee not found");
    } else {
        response.getWriter().println(getJSONedEmployee(employee));
    }
}

public String getJSONedEmployee(Employee e) {
    JSONObject employeeObj = new JSONObject();
    employeeObj.put("firstName", e.getFirstName());
    employeeObj.put("lastName", e.getLastName());
    employeeObj.put("salary", e.getSalary());
    employeeObj.put("vacationHours", e.getVacationHours());

    JSONObject responseObj = new JSONObject();
    responseObj.put("employee", employeeObj);
    return responseObj.toString();
}
}

```

---

### Behind the scenes:

The key elements in this servlet are as follows:

- The `@EJB` annotation provides an instance of the employee facade session bean's business interface (`EmployeeFacadeInterface`). This is the injection technique of EJB 3.0, much simpler than using JNDI:

```
@EJB
```

```
EmployeeFacadeInterface employeeFacade;
```

- The application code required is the same for both a `doPost` and a `doGet`, so we redirect all `doGet` requests to the `doPost` method:

```

protected void doGet(HttpServletRequest request, HttpServletResponse
                    response) throws ServletException, IOException {
    doPost(request, response);
}

```

- The `doPost` method uses the input parameter to look up the employee record from the `EmployeeFacade` session facade. It then passes the `employee` object to the `getJSONedEmployee` method.

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
{

    String empId = (String) request.getParameter("empId");

    Employee employee = null;
    try {

        employee=employeeFacade.findEmployeeById(Integer.parseInt(empId));
    } catch (Exception e) {
        e.printStackTrace();
    }

    if (employee == null) {
        response.getWriter().println("Employee not found");
    } else {
        response.getWriter().println(getJSONedEmployee(employee));
    }
}
```

- The `getJSONedEmployee` method serializes the contents of the `employee` object by using JSON4J library APIs.

```
public String getJSONedEmployee(Employee e) {
    JSONObject employeeObj = new JSONObject();
    employeeObj.put("firstName", e.getFirstName());
    employeeObj.put("lastName", e.getLastName());
    employeeObj.put("salary", e.getSalary());
    employeeObj.put("vacationHours", e.getVacationHours());

    JSONObject responseObj = new JSONObject();
    responseObj.put("employee", employeeObj);
    return responseObj.toString();
}
```

The JavaScript Object Notation (JSON4J) library is an implementation of a set of JavaScript Object Notation (JSON) handling classes for use within Java environments. The JSON4J library provides the following functions:

- ▶ A simple Java model for constructing and manipulating data to be rendered as the JSON implementation.
- ▶ A fast transform for XML to JSON conversion, for situations where conversion from an XML reply from a Web service into a JSON structure is desired for easy use in Asynchronous JavaScript and XML (Ajax) applications. The advantage of this is that Ajax-patterned applications can handle JSON formatted data without having to rely on ActiveX objects in Microsoft Internet Explorer XML transformations and other platform-specific XML parsers. In addition, JSON-formatted data tends to be more compact and efficient to transfer.
- ▶ A JSON string and stream parser that can generate the corresponding JSONObject, which represents that JSON structure in Java. You can then make changes to that JSONObject and serialize the changes back to the JSON implementation.

It was included in the project's classpath when the Web 2.0 facet was enabled. The jar file is not in the WEB-INF/lib directory, it is in the WebSphere Application Server server classpath after the Web 2.0 feature pack was installed.

If you plan to deploy the EAR file into a server that does not have the Web 2.0 feature pack installed you must add the JSON4J library into the WEB-INF/lib directory. The JSON4J jar file is under the server runtime:

`WAS_HOME/web2fep/optionalLibraries/JSON4J/JSON4J.jar`

## 16.7.2 Test the FindEmployee REST servlet

To test the servlet, we deploy it to the server and run the servlet:

1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Select **FindEmployeeServlet** (in DonateWeb20/DonateWeb20/Servlets), right-click and select **Run as** → **Run on Server**.
3. At the Run On Server pop-up:
  - a. Select **ExperienceJEE Test Server**.
  - b. Enable **Always use this server when running this project**.
  - c. Click **Finish**.

4. A Web browser is opened and displays the message 'Employee not found' This is because we have not added the parameter empId to select an employee.
5. Update the url to the following and enter/refresh:

<http://localhost:9080/DonateWeb20/FindEmployeeServlet?empId=2>

And as expected, the browser displays the serialized details of the Employee object in JSON format as follows:

```
{{"employee":{"lastName":"Armstrong","vacationHours":44,"salary":null,"firstName":"Neil"}}
```

### Behind the scenes

When the web 2.0 facet was enabled, several files of dojo api were included in the project. It makes the publishing of the project much slower than the other projects publishing, because the workbench needs to copy all those files into the server

## 16.7.3 Create the Find Employee view

In this section we create the HTML and JavaScript code to invoke the FindEmployeeServlet and display the employee details.

### Create the index.html file

First, we create an index.html file for the starting page:

1. Select **WebContent** (in DonateWeb20), right-click and select **New** → **Web Page**.
2. At the New Web Page pop-up:
  - a. Set the template to **Basic Templates** → **HTML/XHTML**.
  - b. Set file name to **index.html**.
  - c. Click **Finish**.
3. At the resulting index.html editor, replace the contents of the file with the contents of the **F16.4 Web 2.0 Initial Home Page** snippet (Example 16-5).
  - a. Save and close index.html.

*Example 16-5 F16.4 Web 2.0 Initial Home Page snippet*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Web 2.0 Client</title>
```

```

<link rel="stylesheet" type="text/css"
      href="dojo/dijit/themes/tundra/tundra.css" media="all">
<link rel="stylesheet" type="text/css"
      href="dojo/dojo/resources/dojo.css"
media="all">
<script type="text/javascript" src="dojo/dojo/dojo.js"
      djConfig="parseOnLoad: true"></script>
<script type="text/javascript" src="functions.js"></script>
</head>
<body class="tundra">
<form id="form1">
  <input type="text" id="empNumId"
        size="3"
        value="2"
        dojoType="dijit.form.NumberTextBox"
        constraints="{min:1,max:8,places:0}"
        required="true"
  />
  <button dojoType="dijit.form.Button" id="findEmpButton"
        onclick="displayEmployeeDetails">Find</button>

  <div preload="true" dojoType="dijit.layout.ContentPane"
        id="EmployeePane" href="empDetails.html" style="display:none;">
  </div>
</form>
</body>
</html>

```

---

### Behind the scenes:

The key elements in this HTML file are as follows:

- ▶ There are two references to two JavaScript files:
  - dojo.js:** The main script file of dojo client runtime in the dojo folder
  - functions.js:** The JavaScript file that we will use for our client code
- ▶ There are three dojo components in the HTML form:
  - dijit.form.NumberTextBox: Input field for employee number
  - dijit.form.Button: Push button for find employee
  - dijit.layout.ContentPane: Pane for employee details

- ▶ We use the **NumberTextBox** component for entering the employee Id. In this example, we use the **constraints** attribute to specify the maximum and minimum values that are valid for employee Id. We have also specified that this field is mandatory by using the **required** attribute and the **value** attribute can be used to specify the default value.
- ▶ The **Button** component invokes the `displayEmployeeDetails` JavaScript method that we provide in a moment.
- ▶ The **ContentPane** component can be associated with the `div` element and in this example is bound to an HTML file, `empDetails.html`, which has markup for displaying the employee details. The **style** attribute used in this example with the value of `display:none` signifies that this pane is not visible when the HTML file is rendered.

For more information about dojo components, refer to the:

<http://api.dojotoolkit.org/>

## Create the `empDetails.html` file

Next, we create an `empDetails.html` file for the employee details:

First, we create an `index.html` file for the starting page:

1. Select **WebContent** (in `DonateWeb20`), right-click and select **New** → **File**.
2. At the New File pop-up:
  - a. Set File name to **empDetails.html**.
  - b. Click **Finish**.

### Behind the scenes:

We use the New File wizard because this wizard uses the default encoding and this avoids the Conflict in Encoding message that would result if you used the New Web Page wizard along with the F16.5 Web 2.0 Employee Details Page snippet.

The New Page wizard generates a page with the ISO-8859 encoding, both in the properties and with a meta-data statement in the default page template:

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

However, if replaced the Web page template with the F16.5 Web 2.0 Employee Details Page snippet (which does not have a meta-data tag), when saving the file we would receive a Conflict in Encoding pop-up warning that the content encoding (cp1252 on Windows) conflicts with the property encoding (ISO-8859).

There are several alternatives to eliminate this pop-up:

- ▶ Add an encoding statement to the snippet.
- ▶ Change the encoding property manually on the page
- ▶ Generate the artifact as a File (and the file is created using the default encoding).

Adding a meta-data tag is appropriate if the page is a standalone entity, but in this situation it is included under Index,.html and it is better to have index.html set the encoding.

Therefore, we use the New File wizard.

3. At the resulting index.html editor, replace the contents of the empDetails.html file with the contents of the **F16.5 Web 2.0 Employee Details Page** snippet, which has markup language for the first name, last name, salary, and vacation of an employee (Example 16-6).
  - a. Save and close empdetails.html.

#### *Example 16-6 F16.5 Web 2.0 Employee Details Page*

```
<div style="color: navy; font-family: Arial; font-weight: bold; font-size: 10pt">
<table>
  <tr>
```

```

        <td>First Name:</td><td width="5px"></td>
        <td><div id="firstName"></div></td>
    </tr>
    <tr>
        <td>Last Name:</td><td width="5px"></td>
        <td><div id="lastName"></div></td>
    </tr>
    <tr>
        <td>Salary:</td><td width="5px"></td>
        <td><div id="salary"></div></td>
    </tr>
    <tr>
        <td>Vacation:</td><td width="5px"></td>
        <td><div id="vac"></div></td>
    </tr>
</table>
</div>

```

---

## Develop the client logic: functions.js

The retrieve of an employee is developed as a JavaScript function:

1. Select **WebContent** (in DonateWeb20), right-click and select **New** → **Other..**
2. At the New pop-up, select **JavaScript** → **JavaScript Source File** and click **Next**.
3. In the New JavaScript File dialog, set file name to **functions.js** and click **Finish**.
4. At the resulting functions.js editor, replace the contents of the file with the contents of the **F16.6 Web 2.0 Client JavaScript Retrieve Employee** snippet (Example 16-7).
  - i. Save and close functions.js.

*Example 16-7 F16.6 Web 2.0 Client JavaScript Retrieve Employee snippet*

---

```

dojo.require("dojo.parser");
dojo.require("dijit.layout.ContentPane");
dojo.require("dijit.form.NumberTextBox");
dojo.require("dijit.form.Button");

function displayEmployeeDetails() {
    var emp = dijit.byId('empNumId').getValue();

    var deferred = dojo.xhrGet(
        {
            url: "/DonateWeb20/FindEmployeeServlet?empId="+emp,
            handleAs: "json",
            timeout: 5000,

```



```

        preventCache: true,
        form: "form1",
        load: function(response, ioArgs) {
            showEmployeeDetails(response.employee);
        },
        error: function(response, ioArgs) {
        }
    }
}
);
}

function showEmployeeDetails(employee) {
    dojo.style(dijit.byId("EmployeePane").domNode, "display", "block");
    document.getElementById("firstName").innerHTML=employee.firstName;
    document.getElementById("lastName").innerHTML=employee.lastName;
    document.getElementById("salary").innerHTML=employee.salary;
    document.getElementById("vac").innerHTML=employee.vacationHours;
}

```

### Behind the scenes:

The key elements of the `functions.js` file are as follows:

- ▶ **dojo.require** statements, and two JavaScript functions:
  - **displayEmployeeDetails**: This function is executed when the `findEmpButton` button is clicked, because its name is specified as the value of the `onclick` event in the `index.html` file.
  - **showEmployeeDetails**: This function is executed by the `displayEmployeeDetails` function. The employee JSON object is passed to this function and we use the values of this employee object to update the **innerHTML** property of `<div>` elements in `empDetails.html`. In this method, we also make the employee details pane visible by changing the value of **display** style property to **block**.
- ▶ In the **displayEmployeeDetails** function, we first retrieve the value of the entered employee Id and assign it to the JavaScript variable `emp`:
 

```
var emp = dijit.byId('empNumId').getValue();
```
- ▶ Then we use the **dojo.xhrGet** function, which is dojo's implementation of the `XMLHttpRequest` call for the GET operation. This `dojo` function sends an HTTP GET request to the server.

- In our example, we have used the following properties for the `arg` parameter of the `dojo.xhrGet` method:
  - **url**: This property is of type `String` and contains the URL to the server endpoint. In our example, we have specified the URL to the `FindEmployee` REST servlet, and we are also passing the value of the entered employee Id as a URL argument:
 

```
url: "/DonateWeb20/FindEmployeeServlet?empId= +emp
```
  - **load**: This property specifies the callback function that is called on a successful HTTP response code. In our example, we have specified to execute the `showEmployeeDetails` method.
  - **error**: This property specifies the function that is called when the xhr request fails either due to network or server, or invalid URL. It is also called if the load or handle callback function throws an exception.
  - **handleAs**: This property specifies how the response string should be handled or parsed. The acceptable values are `text` (default), `json`, `json-comment-filtered`, `javascript`, and `xml`.
  - **form**: This property specifies the DOM node for the HTML form. It is used to extract form values and send to the server.

## 16.7.4 Test the Find Employee Use Case

To test the `index.html`, we publish the project to the server and run the `index.html` file on the server:

1. Publish the `DonateEAR` enterprise application (in the `Servers` view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Open a Web browser and enter the following URL:
 

```
http://localhost:9080/DonateWeb20/
```
3. By default, you will see that **2** is entered in the employee Id, based on the code in `index.html`. You can this value, or enter another value between 1 and 8 (but only use 1 to 3 if you skipped step 4 on page 570).

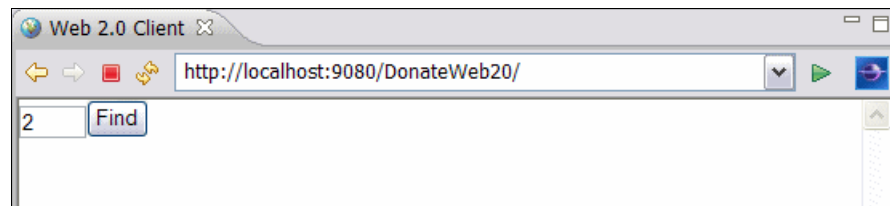


Figure 16-12 *DonateWeb20 Web page*

- Click **Find** to execute the `dojo.xhrGet` function that submits the employee Id to the server, waits for the JSON response, and then executes the `show` function to display the employee details content pane.

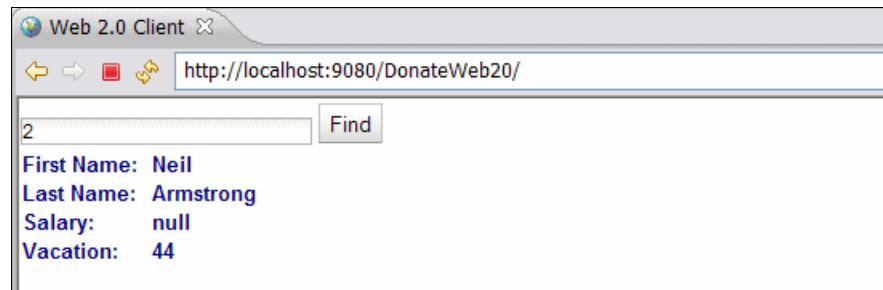


Figure 16-13 Find results

**Behind the scenes:** the salary field will return a null if implemented the programmatic role based access in 10.6.2, “Implement programmatic security in the EmployeeFacade” on page 394.

- Type an invalid value or a value that is not in the range, and verify that the validation is working for the employee Id field.

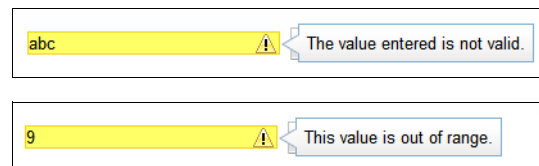


Figure 16-14 Validation results

## 16.8 Add style to the Web 2.0 application

In this section, we add style definitions to the view. We want to provide a background color to the page and also a header with IBM logo and a banner with the book title and a skin for both the content areas of the two use cases, Find Employee and Donate Vacation. We also want to provide each content area with a title (Figure 16-15 on page 584).



Figure 16-15 Adding style to the view

## 16.8.1 Create a cascading style sheet

Style templates are defined in a cascading style sheet that is added to the Web application:

1. Select **WebContent** (in DonateWeb20), right-click and select **New** → **Other**.
2. At the New pop-up, select **Web** → **CSS**, and click **Next**.
3. At the New Cascading Style Sheet pop-up, set File name to **index.css** and click **Finish**.
4. At the resulting index.css editor, replace the contents of the file with **F16.7 Web 2.0 Style File** snippet (Example 16-8).
  - a. Save and close index.css.

Example 16-8 F16.7 Web 2.0 Style File snippet

```
body {
    background: rgb(204, 204, 204) none repeat;
    color: rgb(0, 0, 0); padding: 10px;
}
#pageContent {
    background: rgb(255, 255, 255) none repeat;
    height: auto; width: 760px; margin: auto; margin-bottom: 20px;
    font-family: Verdana,Arial,Helvetica,sans-serif; font-weight: bold;
    font-size:
12px;
}
#header {
```

```

        height: 40px; width: 760px;
        background: rgb(0, 0, 0) none repeat;
    }
    #ibmLogo {
        float: left; margin: 5px; padding: 0px; height: 22px; width: 60px;
        background: url('ibmLogo.gif');
    }
    .bannerTitle {
        float: right; margin: 10px; padding: 0px; height: 22px; width: auto;
        color: rgb(255, 255, 255); font-size: 14px;
    }

    /* Style for the two-column Container*/
    #container #two-columns {
        height: auto; width: 760px;
        margin: auto; margin-bottom: 20px;
    }
    #container #two-columns #column-1 {
        float: left; padding: 5px;
        height: auto; width: 340px;
    }
    #container #two-columns #column-2 {
        float: left; padding: 5px;
        height: auto; width: 340px;
    }

    /* Clear the above float */
    #container .clr { clear: both; }

    /* Style for Form field Labels and Values */
    .fieldLabel { padding: 0px 0px 0px 10px; float: left; width: 120px; }
    .fieldValue { color: rgb(0, 102, 153); }
    .fieldInp { padding: 0px 0px 0px 10px; float: left; }

    /* Style for Portlet Skin */
    #portletTitle {
        padding: 5px 5px 0px; border-width: 0px;
        font-size: 14px; color: rgb(0, 0, 0);
        background: rgb(255, 255, 255) url('portletHeader.gif');
        height: 25px;
    }
    .skinOutline {
        border: 1px solid rgb(104, 104, 104); margin: 10px; padding: 0px;
        width: 320px; height: 250px;
    }
}

```

---

5. Table 16-3 on page 586 describes the style definitions we have introduced in the index.css style sheet.

Table 16-3 Description of the style sheet

Style	Description
body	Gray background for the page.
pageContent	White background for the page content area of 760 pixels. Also specifies the default font family size and weight for the content.
header	A header area at the top of the page with black background color and height of 40 pixels.
ibmLogo	Style that has ibm logo image (ibmLogo.gif) as its background in the header.
bannerTitle	Style for the title of the book in the Banner that is right-aligned in the header.
container, column-1, column-2	Style definitions to provide two columns in the main content area.
fieldLabel, fieldValue, fieldInp	Style definitions for form field labels and their values.
portletTitle and skinOutline	Style definition to provide portlet look and feel with a skin. The header of the portlet has a background that is an image (portletHeader.gif).

6. Figure 16-16 and Figure 16-17 on page 587 illustrate graphically the significance of each style when applied to the Find Employee use case.

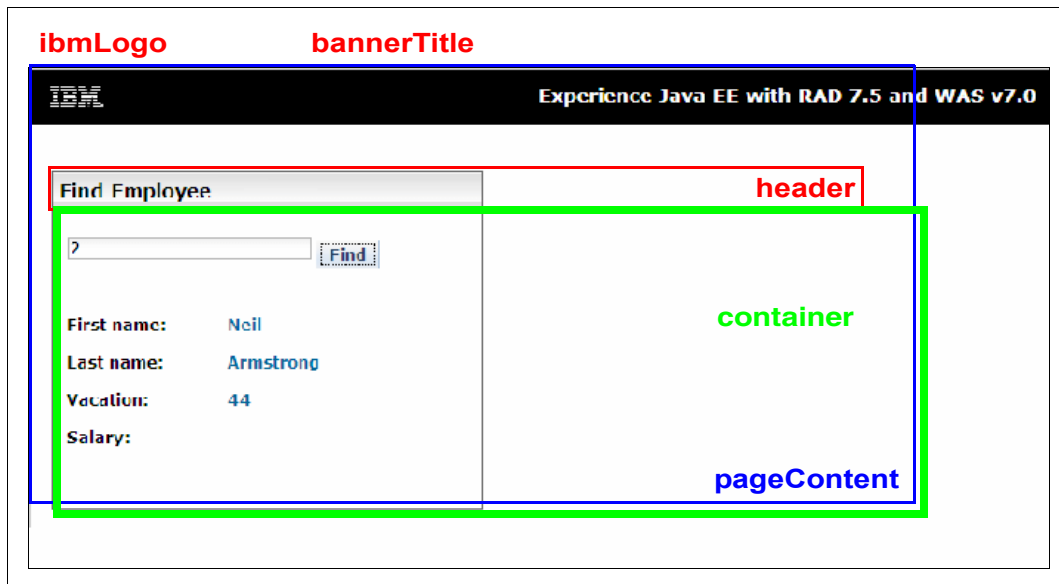


Figure 16-16 Style definitions for Find Employee search

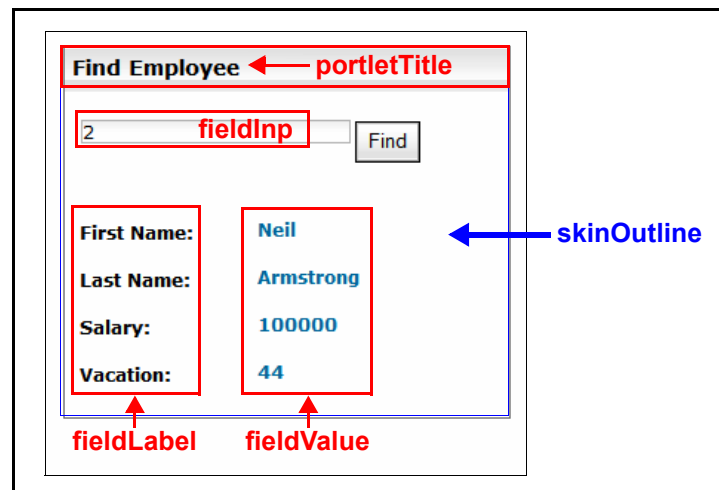


Figure 16-17 Style definitions for Find Employee details

## 16.8.2 Copy the images

The images are provided in the samples ZIP file and must be imported into the workspace.

1. Select **WebContent** (in DonateWeb20), right-click and select **Import**.
2. At the Import /Select pop-up, select **General** → **File System** and click **Next**.
3. At the Import/File system pop-up:
  - a. Set From Directory to **C:\7827code\web2.0**
  - b. In the directory listing on the left, enable **web2.0**
  - c. Click **Finish**.

**Behind the scenes:** the following files are copied into the WebContent folder:

- ▶ `ibmLogo.gif`: The IBM logo in the Header
- ▶ `portletHeader.gif`: Background gradient image for the Portlet titl

## 16.8.3 Add style definitions to the HTML files

The individual HTML pages must be modified to reference the new style sheet and the associated image files.

1. Replace the contents of **index.html** (in DonateWeb20/WebContent) with the contents of the **F16.8 Web 2.0 Add Style Home Page** snippet (Example 16-9 on page 588).

*Example 16-9 F16.8 Web 2.0 Add Style Home Page snippet*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Web 2.0 Client</title>
<link rel="stylesheet" type="text/css"
href="dojo/dijit/themes/tundra/tundra.css"
    media="all">
<link rel="stylesheet" type="text/css" href="dojo/dojo/resources/dojo.css"
    media="all">
<link rel="stylesheet" type="text/css" href="index.css" media="all">

<script type="text/javascript" src="dojo/dojo/dojo.js"
    djConfig="parseOnLoad: true"></script>
<script type="text/javascript" src="functions.js"></script>
</head>
<body class="tundra">
```



```

<form id="form1">
  <div id="pageContent">
    <div id="header">
      <div id="ibmLogo"></div>
      <span class="bannerTitle">
        Experience Java EE with WAS v7.0 and RAD 7.5</span>
    </div>
    <div id="container">
      <div id="two-columns">
        <div id="column-1">
          <div class="skinOutline">
            <div id="portletTitle">Find Employee</div>
            <p>
              <span class="fieldInp">
                <input type="text" id="empNumId"
                  size="3"
                  value="2"
                  dojoType="dijit.form.NumberTextBox"
                  constraints="{min:1,max:8,places:0}"
                  required="true"
                />
              </span>
              <button dojoType="dijit.form.Button" id="findEmpButton"
                onclick="displayEmployeeDetails">Find</button>
            </p>
            <div preload="true" dojoType="dijit.layout.ContentPane"
              id="EmployeePane" href="empDetails.html"
              style="display:none;">
          </div>
        </div> <!-- end of column-1 -->
        <div class="clr"></div>
      </div> <!-- end of two-columns -->
    </div>
  </div>
</form>
</body>
</html>

```

---

2. Replace the contents of **empDetails.html** (in DonatWeb20/WebContent) with the contents of the **F16.9 Web 2.0 Add Style Employee Details** snippet (Example 16-10).

*Example 16-10 F16.9 Web 2.0 Add Style Employee Details snippet*

---

```

<br>
  <label class="fieldLabel">First name:</label>
  <div id="firstName" class="fieldValue"></div>
<br>
  <label class="fieldLabel">Last name:</label>

```

```
<div id="lastName" class="fieldValue"></div>
<br>
<label class="fieldLabel">Vacation:</label>
<div id="vac" class="fieldValue"></div>
<br>
<label class="fieldLabel">Salary:</label>
<div id="salary" class="fieldValue"></div>
```

---

## 16.8.4 Test the Find Employee use case with style

To test again, we publish the project to the server and run the index.html on the server:

1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Open a Web browser and enter the following URL:  
<http://localhost:9080/DonateWeb20/>
3. The home page opens. Type an employee ID (2) and click **Find**. The employee is retrieved and the employee details are displayed (Figure 16-18 on page 590).



Figure 16-18 Run of the Find Employee use case with style definitions

## 16.9 Implement the Donate Vacation use case

In this section we implement the Donate Vacation use case, where an employee donates vacation hours to a donation fund.

### 16.9.1 Create the view for vacation donation

In this section we create the DonatePane view for the Donate Vacation use case

1. Insert the contents of the **F16.10 Web 2.0 Add Style Home Page Second Column** snippet into **index.html** (in DonateWeb20/WebContent) after the end of the first column (Example 16-11).
  - a. Save and close index.html.

*Example 16-11 F16.10 Web 2.0 Add Style Home Page Second Column snippet*

---

```
</div> <!-- end of column-1 -->          <=== INSERT AFTER THIS LINE
<div id="column-2">
  <div class="skinOutline">
    <div id="portletTitle">Donate Vacation</div>
    <div preload="true" dojoType="dijit.layout.ContentPane"
      id="DonatePane" href="donate.html"
style="display:none;">
    </div>
  </div>
</div>
<div class="clr"></div>          <=== INSERT BEFORE THIS LINE
```

---

#### Behind the scenes:

In this code snippet, we add style definitions to add a second column to `index.html`. In the second column we add another **dijit.layout.ContentPane** component and bind it to a new **donate.html** file.

As we did for the Find Employee use case, we add markup for the skin to surround the content pane and also provide a title (Donate Vacation). Also notice that we set the value of the style attribute to **display:none**, which means that this content pane is not visible when `index.html` is rendered initially. We will make it visible when the user clicks **Find** to retrieve details of the employee.

### Display the Donate pane in the second column

In this section we activate the DonatePane after an employee is retrieved.

1. In the showEmployeeDetails function in **functions.js** (in DonateWeb20/WebContent) add one line to show the DonatePane.
  - a. Save and close functions.js.

```
function showEmployeeDetails(employee) {
    dojo.style(dijit.byId("EmployeePane").domNode, "display", "block");
    dojo.style(dijit.byId("DonatePane").domNode, "display", "block");
    document.getElementById(.....
    .....
```

## Create the donate.html file

Next, we create the donate.html file for the Donate Vacation use case.

1. Select **WebContent** (in DonateWeb20), right-click and select **New** → **File**.
2. At the New File pop-up:
  - a. Set File name to **donate.html**.
  - b. Click **Finish**.
3. At the resulting donate.html editor, replace the contents of the empDetails.html file with the contents of the **F16.11 Web 2.0 Donate Vacation Page** snippet, which has markup for selecting the fund name, the amount of vacation to donate to that fund, and the Donate button (Example 16-12 on page 592).
  - a. Save and close donate.html.

*Example 16-12 F16.11 Web 2.0 Donate Vacation Page snippet*

---

```
<div dojoType="dojo.data.ItemFileReadStore" jsId="fundsStore"
url="funds.json"></div>
<p/>
  <label class="fieldLabel">Fund name:</label>
  <input id="fundNameField" dojoType="dijit.form.FilteringSelect"
    store="fundsStore"
    searchAttr="name"
    name="fundName"
    autocomplete="true"
    invalidMessage= "Please select a valid donation Fund"
  />
<p/>
  <label class="fieldLabel">Number of hours:</label>
  <input type="text" id="vacField"
    name="hours" size="3"
    value="1"
    dojoType="dijit.form.NumberTextBox"
    promptMessage="Enter number of hours of vacation to donate"
    constraints="{min:1,max:90,places:0}"
```

```

        invalidMessage= "Please enter a valid number of hours"
        required="true"
    />
</p>
    <span class="fieldInp">
        <button dojoType="dijit.form.Button" id="donateButton"
onclick="donate">Donate</button>
    </span>

```

---

### Behind the scenes:

In this code snippet, we add markup for two input form fields where the user can select a fund and the number of their vacation hours to donate to that fund. There is also a markup for a Donate button for the user to click to submit the donation request.

The key elements in this HTML file are as follows:

- **ItemFileReadStore:** In this code snippet, we use a data store called `ItemFileReadStore`, that comes pre-packaged with Dojo:

```

<div dojoType="dojo.data.ItemFileReadStore" jsId="fundsStore"
url="funds.json"></div>

```

And as the name suggests, the `ItemFileReadStore` is a read-only data store that reads the contents from an HTTP endpoint. This data store expects the contents of this source to be in JSON format, which it reads and deserializes into an in-memory JavaScript object. The HTTP endpoint is specified as the value of the `url` attribute. This URL can point either to a static file with JSON content or to a dynamic service hosted at the server. This service can either be a servlet that writes text in JSON format to the output stream or a JAX-WS based REST Web Service.

In this case, the contents of the `funds.json` file are read and deserialized into an in-memory JavaScript object that is specified as the value of the `jsId` attribute `fundsStore`.

- **FilteringSelect:** In this code snippet, we use a `dijit.form` component that is just like an HTML select or drop-down form field, but is populated dynamically using the contents of the in-memory JavaScript object that is specified as the value of the `store` attribute:

```
<input id="fundNameField" dojoType="dijit.form.FilteringSelect"
      store="fundsStore"
      searchAttr="name"
      name="fundName"
      autocomplete="true"
      invalidMessage= "Please select a valid donation Fund"
/>
```

In this code snippet, we use the contents of the `fundsStore` object that was created by the `ItemFileReadStore` by reading the contents of the `funds.json` file. Or in other words, we are pre-populating the contents of the fund name drop-down field with the contents of a static JSON file.

- We are using the **NumberTextBox** component for specifying the number of hours to donate. We use the `constraints` attribute to specify the maximum and minimum values that are valid. We also specify that this field is mandatory by using the `required` attribute, and the `value` attribute can be used to specify the default value.
- The **Button** component in this example results in the invocation of the `donate` JavaScript method that we will implement in a moment.

## Create the funds.json file

Next, we create the static `funds.json` file that holds the names of the donation funds:

1. Select **WebContent** (in `DonateWeb20`), right-click and select **New** → **File**.
2. At the New File pop-up, type **funds.json** as the file name and click **Finish**.
3. Replace the contents of the new `funds.json` file with the contents of the **F16.12 Web 2.0 Static Fund Names JSON file** snippet, which has all the fund names in JSON format (Example 16-13).
  - a. Save and close `funds.json`.

*Example 16-13 F16.12 Web 2.0 Static Fund Names JSON file snippet*

```
{identifier:"name",
 items: [
   {name:"DonationFund"},
   {name:"Red Cross"},
```

```
{name:"United Way"},
{name:"Heart and Stroke Foundation"} <==== does not exist in database
]}
```

---

#### Behind the scenes:

- The `identifier` attribute instructs the data store indicating which field contains the value to be submitted, in our case the `name` field. In this JSON snippet for funds, the `items` object only has one attribute, but it could have multiple attributes.

For example, in the following JSON snippet for US states, we can use one attribute (`label`) to display the name and another attribute (`abbreviation`) to submit the value (as set by the `identifier`):

```
{identifier:"abbreviation",
 items: [
   {name:"Alabama",label:"Alabama",abbreviation:"AL"},
   ...
 ]}
```

- Notice that one fund name does not match the fund names stored in the Donate database. This allows for testing of an invalid name.

At a later stage, in 16.9.7, “Retrieve the fund names from the database” on page 606, we replace the static `funds.json` file with a dynamic JSON object that is retrieved from the Donate database through a servlet.

## 16.9.2 Implement the client logic for the Donate Vacation use case

The client logic that is executed when the **Donate** button is clicked is added to the **functions.js** JavaScript file in the donate function:

1. Add the following statements at the bottom of **functions.js** (in `DonateWeb20/WebContent`).

```
dojo.require("dojo.data.ItemFileReadStore");
dojo.require("dijit.form.FilteringSelect");
dojo.require("dijit.Dialog");
```

#### Behind the scenes:

Because we introduced two new dojo components, `ItemFileReadStore` and `FilteringSelect`, we have to add `require` statements to the JavaScript file. To display success and error messages in a dialog box, we also require the `Dialog` component

2. Insert the contents of the **F16.13 Web 2.0 Implement JavaScript Client Logic** snippet at the end of the **functions.js** (in DonateWeb20/WebContent) (Example 16-14).
- a. Save and close **functions.js**.

*Example 16-14 F16-.3 Web 2.0 Implement JavaScript Client Logic snippet*

---

```
function donate() {
    dijit.byId("donateButton").disabled=true;
    var emp = dijit.byId('empNumId').getValue();
    var fund = dijit.byId('fundNameField').getValue();
    var hours = dijit.byId('vacField').getValue();

    var deferred = dojo.xhrGet(
        {
            url: "/DonateWeb20/DonateServlet?employeeId="+emp
                +"&fundName="+fund+"&hours="+hours,
            handleAs: "json",
            timeout: 5000,
            preventCache: true,
            form: "form1",
            load: function(response, ioArgs) {
                showMessageDialog(response);
                dijit.byId("donateButton").disabled=false;
                return response;
            },
            error: function(response, ioArgs) {
                document.getElementById('errMsg').innerHTML =
                    "Error during xhrPost for
Donate";
                dijit.byId('errDialog').show();
                return response;
            }
        }
    );
}
```

---



## Behind the scenes:

The key elements of the donate function are as follows:

- Disable the Donate button to avoid double-clicking:

```
dijit.byId("donateButton").disabled=true;
```

We reactivate this button after the request processing is complete.

- Get the values that the user entered or selected in the employee number, fund name, and number of hours form fields:

```
var emp = dijit.byId('empNumId').getValue();
var fund = dijit.byId('fundNameField').getValue();
var hours = dijit.byId('vacField').getValue();
```

- Execute a REST servlet, DonateServlet, by sending an HTTP GET request to the server using the `dojo.xhrGet` API and submitting the values of employee number, fund name, and the number of hours as arguments:

```
var deferred = dojo.xhrGet(
{
    url: "/DonateWeb20/DonateServlet?employeeId="+emp
   +"&fundName="+fund+"&hours="+hours,
    handleAs: "json",
    .....
    load: function(response, ioArgs) {
        .....
    },
    error: function(response, ioArgs) {
        .....
    }
}
);
```

- In the callback function specified as the value of the **load** attribute of the `dojo.xhrGet` API, we pass the result string we get from the DonateServlet (in JSON format) to a `showMessageDialog` function that we create in a moment. We also enable the Donate button that we disabled earlier.

```
load: function(response, ioArgs) {
    showMessageDialog(response);
    dijit.byId("donateButton").disabled=false;
    return response;
},
```

- We have also specified an **error** call back function, which is called if there is an error executing the `dojo.xhrGet` API. In this function, we set an error message string as the value of the `innerHTML` property of the `errMsg` element. We also call the `show` method of the `errDialog` dijit component. This action displays the specified error message using dojo's message dialog component. We will specify the markup of the `errMsg` and `errDialog` elements in a moment.

```
error: function(response, ioArgs) {
    document.getElementById('errMsg').innerHTML =
        "Error during xhrPost for Donate";
    dijit.byId('errDialog').show();
    return response;
}
```

### 16.9.3 Display messages in Ajax modal dialogs

We have a requirement to display the success and error messages in Ajax modal dialogs. In this section we add the markup to display the Ajax modal dialogs using the `dijit.Dialog` component:

1. Insert the contents of the **F16.14 Web 2.0 Markup for Ajax Modal Dialog Messages** snippet into **index.html** (in `DonateWeb20/WebContent`) after the end of second column (Example 16-15).
  - a. Save and close `index.html`.

*Example 16-15 F16.14 Web 2.0 Markup for Ajax Modal Dialog Messages snippet*

```
</div> <!-- end of two-columnns --> <=== INSERT AFTER THIS LINE

<!-- Using dijit dialog to show status messages-->
<div dojoType="dijit.Dialog" id="msgDialog" title="Thank you!"
    style="font:11px Arial; display:none; color:navy;
font-weight:bold;">
    <div id="msg" style="font:11px Arial; color:red;
padding:10px;"></div>
</div>

<div dojoType="dijit.Dialog" id="errDialog" title="Error Message"
    style="font:11px Arial; display:none; color:navy;
font-weight:bold;">
    <div id="errMsg" style="font:11px Arial; color:red; padding:10px;">
    </div>
</div>
```

### Behind the scenes:

Note that the `id` attributes for displaying the error message match the `ids` used in the `functions.js` JavaScript:

```
error: function(response, ioArgs) {
    document.getElementById('errMsg').innerHTML = ".....";
    dijit.byId('errDialog').show();
    return response;
}
```

2. In a first simple implementation, we only display the message returned from the server. Insert the **F16.15 Web 2.0 Show Message Dialog Simple** snippet at the end of **functions.js** (in `DonateWeb20/WebContent`) (Example 16-16).
  - a. Save and close **functions.js**.

*Example 16-16 F16.15 Web 2.0 Show Message Dialog Simple snippet*

---

```
function showMessageDialog(result){
    dijit.byId("findEmpButton").disabled=true;
    document.getElementById("msg").innerHTML = result.returnMessage;
    dijit.byId("msgDialog").show();
    dijit.byId("findEmpButton").disabled=false;
}
```

---

### Behind the scenes:

This code disables the Find button, sets the `msg` attribute from the servlet response, displays the message dialog (`msgDialog` matches the `id` in `index.html`), and enables the Find button when the dialog is closed.

## 16.9.4 Create a REST servlet for the Donate Vacation use case

In this section, we create a REST servlet that submits the donation request to the `DonateBean` session EJB through simple EJB injection and access. This servlet then serializes the return message in JSON format and writes that string to the HTTP response output stream. This servlet is called through the `dojo.xhrGet` API in the `donate` function specified in the `functions.js` file:

1. In the Enterprise Explorer, select **DonateWeb20**, right-click and select **New** → **Servlet**.

2. At the Create Servlet pop-up, provide the following data:
  - Java package: **donate.web.rest.servlet**
  - Class name: **DonateServlet**
  - a. Click **Finish**.
3. In the DonateServlet editor, delete all the generated method skeletons (default constructor, doGet, and doPost), leaving the base class definition and the serialVersionUID.
 

```
public class DonateServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

}
```
4. Insert the **F16.16 Web 2.0 Servlet for Donate Vacation Use Case** snippet before the closing brace (Example 16-17).
5. Organize imports (**Ctrl+Shift+O**).
  - a. At the Organize Imports pop-up, select **com.ibm.json.java.JSONObject** and click **Finish**.

**Behind the scenes:** the following imports are resolved:

```
import java.io.IOException;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ibm.json.java.JSONObject;
import donate.ejb.interfaces.DonateBeanRemote
```

6. Save and close DonateServlet.java.

*Example 16-17 F16.16 Web 2.0 Servlet for Donate Vacation Use Case snippet*

---

```
public class DonateServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB
    DonateBeanRemote donate;

    protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request,
```

```

        HttpServletResponse response)
        throws ServletException, IOException {
    int employeeId = Integer.parseInt(request.getParameter("employeeId"));
    String fundName = request.getParameter("fundName");
    int hours = Integer.parseInt(request.getParameter("hours"));
    String result = donate.donateToFund(employeeId, fundName, hours);
    JSONObject resultObj = new JSONObject();
    resultObj.put("returnMessage", result);
    response.getWriter().println(resultObj.toString());
}
}

```

### Behind the scenes:

The DonateServlet is similar to the FindEmployeeServlet. In the doPost method, the parameters are retrieved from the donate.html fields, the donateToFund method of the session EJB is invoked, and the result is returned as a JSON object with one attribute (returnMessage).

## Test the DonateServlet

To test the servlet, we publish the application to the server and run the servlet:

1. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
2. Open a Web browser and enter the following URL:

<http://localhost:9080/DonateWeb20/DonateServlet?employeeId=2&fundName=Red%20Cross&hours=10>

- a. Verify that the browser displays the serialized details of the success message in JSON format:

```
{"returnMessage":"Transfer successful"}
```

3. Run the servlet with an invalid fund name and the result is an error message:

<http://localhost:9080/DonateWeb20/DonateServlet?employeeId=2&fundName=BadFund&hours=2>

```
{"returnMessage":"Error: Fund not found with name \"BadFund\""}
```

## 16.9.5 Test the Donate Vacation use case

Now we can test the Donate Vacation use case:

1. Open a Web browser and enter the following URL:

<http://localhost:9080/DonateWeb20/>

- Retrieve the employee by clicking **Find**. The employee details and the second column are displayed.

The screenshot shows a web application interface with two main panels. The left panel, titled "Find Employee", contains a text input field with the value "2" and a "Find" button. Below this, the employee details are displayed: First name: Neil, Last name: Armstrong, Vacation: 37, and Salary: 100000. The right panel, titled "Donate Vacation", contains a "Fund name:" dropdown menu, a "Number of hours:" input field with the value "1", and a "Donate" button.

Figure 16-19 Employee details

- Experiment with the type-ahead function by typing r into the Fund name field. The field is filled with the correct fund name.

This screenshot is similar to the previous one, but the "Fund name:" dropdown menu in the "Donate Vacation" panel now displays "Red Cross" as the selected option. The "Find Employee" panel remains unchanged, showing the same employee details.

Figure 16-20 Using the type-ahead function

- Alternatively use the pull-down menu to select a fund.

Fund name: **Red Cross**  
DonationFund  
Red Cross  
United Way  
Heart and Stroke Foundation

Number of hours:

**Donate**

Figure 16-21 Using the pull-down function

- Select a fund, type a number of hours to donate, and click **Donate** to transfer the vacation hours to a fund. The dialog box with the successful message is displayed.

IBM Experience Java EE with RAD 7.5 and WAS v7.0

**Find Employee**

2 **Find**

First name: Neil  
Last name: Armstrong  
Vacation: 37  
Salary: 100000

**Donate Vacation**

Fund name: Red Cross  
Number of hours: 1  
**Donate**

**Thank you!**  
Transfer successful

Figure 16-22 Transfer the vacation hours

- Close the message dialog, and click **Find** to verify that the number of vacation hours has been decreased for the employee.

IBM
Experience Java EE with RAD 7.5 and WAS v7.0

### Find Employee

First name: Neil

Last name: Armstrong

Vacation: 36

Salary: 100000

### Donate Vacation

Fund name: Red Cross ▼

Number of hours:

Figure 16-23 Results

7. If you donate hours to the Heart and Stroke Foundation fund, which does not exist in the database, or if you try to donate more hours than are available, you get an error message box.

## 16.9.6 Update the vacation hours automatically

We can improve the application by updating the vacation hours of the employee in the employee details, and to improve the transfer successful message with the remaining hours:

1. In **functions.js** (in `DonateWeb20/WebContent`), rename the `showMessageDiaog` function to `showMessageDialog1`:
 

```
function showMessageDialog1(result){
    .....
}
```
2. Add an improved version of the `showMessage` function to **functions.js** by inserting the **F16.17 Web 2.0 Show Message Dialog Hours** snippet at the end of the file (Example 16-18).
  - a. Save and close `functions.js`.

*Example 16-18 F16.17 Web 2.0 Show Message Dialog Hours snippet*

---

```
function showMessageDialog(result){
    digit.byId("findEmpButton").disabled=true;
    var emp = digit.byId('empNumId').getValue();
    var empObj;
```



```

var deferred = dojo.xhrGet(
{
    url: "/DonateWeb20/FindEmployeeServlet?empId="+emp,
    handleAs: "json",
    timeout: 5000,
    preventCache: true,
    form: "form1",
    load: function(response, ioArgs) {
        empObj = response.employee;
        document.getElementById("vac").innerHTML =
            empObj.vacationHours;
        document.getElementById("msg").innerHTML =
result.returnMessage
            + ". You have " + empObj.vacationHours
            + " hours of vacation remaining";
        dijit.byId("msgDialog").show();
        dijit.byId("findEmpButton").disabled=false;
    },
    error: function(response, ioArgs) {
        console.debug
            ("**** dojo.xhrGet for employee details NOT successful.
");
    }
}
);
}

```

### Behind the scenes:

In the improved implementation we invoked the `FindEmployeeServlet` to retrieve the updated employee information as a JSON object (`empObj`). We store the remaining vacation hours (`empObj.vacationHours`) in the `vac` field of the employee details pane, and we add the remaining hours to the result message (`msg`).

3. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
4. Rerun the Donate Vacation use case as described in “16.9.5 Test the Donate Vacation use case” on page 601. The vacation hours are updated in the employee details pane and the extended message is displayed.



Figure 16-24 Extended message

## 16.9.7 Retrieve the fund names from the database

Instead of using the static file `funds.json`, we want to populate the drop-down box with the fund names from the database. We create a `GetAllFunds` servlet to retrieve the names in JSON format, and change the `donate.html` file to invoke the servlet to populate the names:

1. Open **donate.html** (in `DonateWeb20/WebContent`) and change the first line with the `ItemFileReadStore` to invoke the `GetAllFunds` servlet instead of the current `funds.json` function:

```
<div dojoType="dojo.data.ItemFileReadStore" jsId="fundsStore"
      url="GetAllFunds"></div>
```

2. Select **donate.web.rest.servlet** (in `DonateWeb20/Java Resources/src`), right-click and select **New** → **Servlet**.
3. At the Create Servlet pop-up, set Class name to **GetAllFunds** and click **Finish**.
4. In the `GetAllFunds` editor, delete all the generated method skeletons (default constructor, `doGet`, and `doPost`), leaving the base class definition and the `serialVersionUID`.

```
public class GetAllFunds extends HttpServlet {
    private static final long serialVersionUID = 1L;

}
```

5. Insert the **F16.18 Get All Funds Servlet** snippet before the closing brace (Example 16-19).
6. Organize imports (**Ctrl+Shift+O**).
  - a. At the Organize Imports pop-up, select **com.ibm.json.java.JSONObject**, **java.util.list** and click **Finish**.

**Behind the scenes:** the following imports are resolved:

```
import java.io.IOException;
import java.util.List;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ibm.json.java.JSONObject;
import donate.ejb.interfaces.FundFacadeInterface;
import donate.entities.Fund;
```

7. Save and close `GetAllFunds.java`.

*Example 16-19 F16.18 Get All Funds Servlet snippet*

---

```
public class GetAllFunds extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB
    FundFacadeInterface fundFacade;

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        List<Fund> funds = null;
        try {
            funds = fundFacade.getAllFunds();
        } catch (Exception e) {
            e.printStackTrace();
        }

        if (funds == null || funds.isEmpty()) {
            response.getWriter().println("Funds not found");
        } else {
            response.getWriter().println(getJSONedFunds(funds));
        }
    }
}
```

```

    }

    public String getJSONedFunds(List<Fund> funds) {
        StringBuffer resp = new StringBuffer
            ("{"identifier: \"name\", items: [");
        for (int i=0; i<funds.size(); i++) {
            Fund f = funds.get(i);
            JSONObject fundObj = new JSONObject();
            fundObj.put("name", f.getName());
            String coma="";
            if (i < funds.size()-1)
                coma=",";
            resp.append(fundObj.toString());
            resp.append(coma);
        }
        resp.append("]}");

        //System.out.println("GetAllFunds servlet: " + resp.toString() );
        return resp.toString();
    }
}

```

---

### Behind the scenes:

The GetAllFunds servlet is similar to the FindEmployeeServlet. It uses the FundFacadeInterface of the session bean to retrieve the fund names, and invokes the getJSONedFunds method to create the JSON object to return.

### Extra Credit:

Optionally publish the application and test the servlet using the URL:

<http://localhost:9080/DonateWeb20/GetAllFunds>

The servlet returns the string:

```

{"identifier: "name", items: [{"name": "Test Fund
#1931088537"}, {"name": "Test Fund #241225349"}, {"name": "Test Fund
#1623324356"}, {"name": "Test Fund #2143525368"}, {"name": "Test Fund
#1203895160"}, {"name": "Test Fund
#1690414285"}, {"name": "DonationFund"}, {"name": "Red
Cross"}, {"name": "United Way"}, {"name": "Breast Cancer Society"}]}

```

Note the numeric fund names such as Test Fund #1931088537. These were created while running the DonateBeanTester JUnit test case in 5.8, “Test the Employee and Donate entities” on page 179 and were never deleted.

8. Publish the DonateEAR enterprise application (in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**).
9. Rerun the Donate Vacation use case as described in “16.9.5 Test the Donate Vacation use case” on page 601. Note that the pull-down list now displays the set of funds listed in the Donate table instead of a static list.

The screenshot shows the IBM Experience Java EE with RAD 7.5 and WAS v7.0 interface. It contains two web forms side-by-side.

**Find Employee**

Input field: 2

First name: Neil  
 Last name: Armstrong  
 Vacation: 32  
 Salary:

**Donate Vacation**

Fund name:   
 Number of hours:

The dropdown menu for Fund name is open, showing the following list of funds:

- Test Fund #1931088537
- Test Fund #241225349
- Test Fund #1623324356
- Test Fund #2143525368
- Test Fund #1203895160
- Test Fund #1690414285
- DonationFund
- Red Cross
- United Way
- Breast Cancer Society

Figure 16-25 Fund names from the Donate table

## 16.10 Monitor the Web 2.0 interactions

We can use the TCP/IP Monitor to see the requests that are issued to the server when running the Web 2.0 application:

1. Define and start a TCP/IP Monitor as described in 11.7.1, “Monitor the SOAP traffic using TCP/IP Monitor” on page 437.

**Alternative:** you can also define TCP/IP Monitor from the Servers view by selecting the server, right-click and selecting **Monitoring** → **Properties**.

2. Run the Web 2.0 application using the URL:  
<http://localhost:9081/DonateWeb20/>
3. In the TCP/IP Monitor view, you can see the traffic between the Web 2.0 client and the server, including the calls to the REST servlets.

## 16.11 Explore

In this section we provide additional Explore resources that could be used to enhance the Donate application.

### 16.11.1 Flex

Flex is an Open Source framework, originally developed by Macromedia, which got acquired by Adobe, for building highly interactive, expressive Web applications that can run in the browser using Adobe Flash Player runtime or on the desktop on Adobe AIR runtime. The Adobe Flash plug-in is available for every major browser and because countless Web sites uses Flash technology, the plug-in is pervasive.

### 16.11.2 Macromedia XML (MXML)

MXML is an XML-based markup language that is used to describe user interface layout and behavior. MXML code is used to declare UI components for creating rich user interfaces. These components are part of the rich library of extensible UI components that Flex provides.

For example, you use the `<mx:Button>` tag to create an instance of a button. Example 16-20 on page 610 shows the complete code required to create a Flex application that displays a button control.

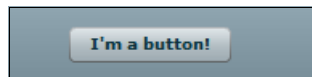
*Example 16-20 Sample MXML file with code that renders a simple button*

---

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Button id="someButton" label="I'm a button!" />
</mx:Application>
```

---

If you have coded the application using other UI markup technologies, such as Java Server Faces (JSF) or JSP tags, then the foregoing code snippet will be quite easy to understand. After you write a Flex application, you have to compile it using the Flex compiler and subsequently execute it to display the result on an HTML page (Figure 16-26).



*Figure 16-26 UI Rendering of a Flex button component*

### 16.11.3 ActionScript

ActionScript is an object-oriented programming language that is used to implement client side logic. ActionScript is based on ECMAScript, the same standard that is also the basis for JavaScript. ActionScript is the programming language for the Adobe Flash Player run-time environment. It enables interactivity, data handling, and much more in Flash content and applications.

ActionScript is executed by the ActionScript Virtual Machine (AVM), which is part of Flash Player. ActionScript code is typically compiled into byte code format by a compiler. The byte code is embedded in SWF files, which are executed by the Flash Player run-time.

In Example 16-21, the HelloWorld class includes a single typeHelloWorld method, which returns a string that says Hello World!. This code snippet would be quite easy to understand for Java and JavaScript programmers.

*Example 16-21 HelloWorld class written in ActionScript v3.0 syntax*

---

```
public class HelloWorld {  
    public function typeHelloWorld():String {  
        var hwString:String;  
        hwString = "Hello World!";  
        return hwString;  
    }  
}
```

---

Adobe Flex Builder (Figure 16-27 on page 612) is a commercial Eclipse based Integrated Development Environment (IDE) tool that facilitates intelligent coding, interactive step-through debugging, and visual design of the user interface layout, appearance, and behavior of rich Internet applications.

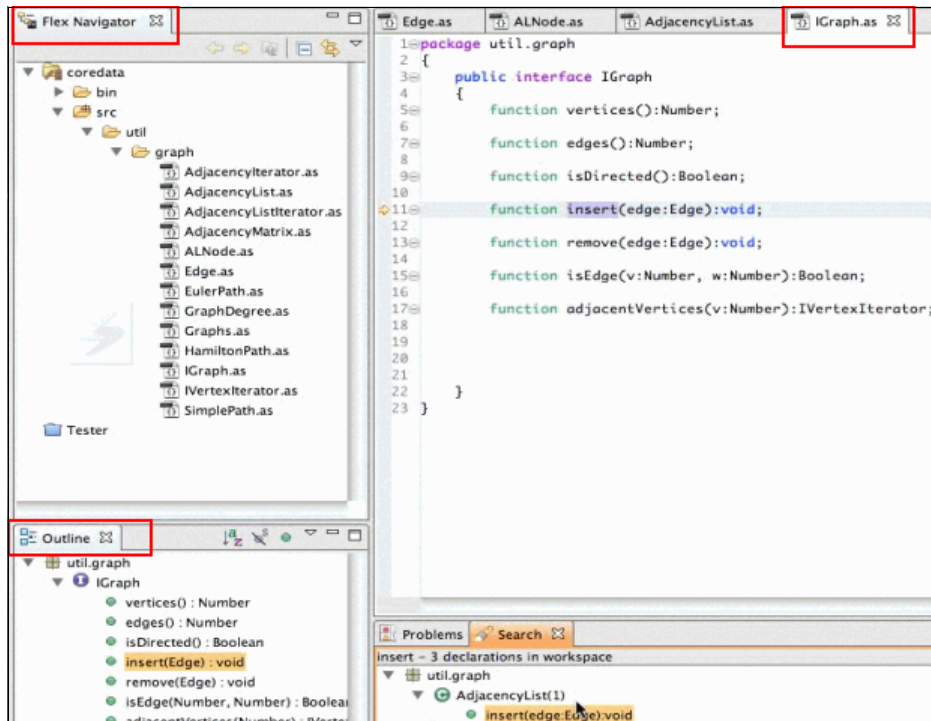


Figure 16-27 Adobe Flex Builder IDE





## Part 6

# Appendixes

This part contains information on how to obtain the download material for the book. It also provides “jump start” information for readers that would like to use the sample instructions but are not performing the samples sequentially throughout the book.





## Jump start

This appendix provides the instructions needed to *jump start* individual chapters, as described in Chapter 1, “Introduction” on page 3.

Jump-start can be defined as follows: *to start or restart rapidly or forcefully*

From Merriam-Webster on-line dictionary: <http://www.merriam-webster.com>

The intention is that most users will implement the Experience sections of this book in a sequential manner: Starting with the chapters in Part 1, “Preliminary activities” on page 1 and following in order.

However, readers who are already familiar with parts of Java EE might want to skip chapters or access them in a random manner. For those readers, this book provides this jump start section to allow readers to quickly configure Rational Application Developer workbench, the embedded WebSphere Application Server test server, and the Rational Application Developer workspace each time they access a chapter out of or order or randomly.

## A.1 Common jump start activities

These steps should be completed for all jump starts:

1. Complete all the activities in Chapter 2, “Install and configure software” on page 25.

These are one-time steps and do not need to be repeated if they have already been completed.

2. Complete all the activities in Chapter 3, “Configure the development environment” on page 53.

### Reminder!

The ExperienceJEE Test Server and the Derby Network Server must be running to complete the subsequent steps in this appendix.

3. Complete all the activities in Chapter 4, “Prepare the legacy application” on page 85.

These are one-time steps and do not need to be repeated if they have already been completed.

4. If you have already completed some chapters in the book, you have to clear the workspace:

- a. In the Servers view, select **ExperienceJEE Test Server**, right-click and select **Add and Remove Projects**.

- i. At the Add and Remove Projects pop-up, select **DonateEAR** in the right pane, and click **Remove** to move it to the left pane.
- ii. Click **Finish**.

**Extra Step?** Also remove **DonateWSCClientEAR** if it is published to the server.

- b. Back in the Servers view, the publishing task occurs in the background, with the status displayed in the lower right task bar. Wait for the task to complete and the server status changes to Synchronized.

**Off course?** If the deployment does not occur automatically, manually initiate the publishing operation (select **ExperienceJEE Test Server**, right-click and select **Publish**.)

- c. In the Enterprise Explorer, ensure that all the projects are contracted (not expanded).
- d. In the Enterprise Explorer, select all projects, right-click and select **Delete**.
- e. At the **Delete Resources** pop-up, select **Delete project contents on disk (...)** click **OK**.

### Off course?

You may receive a Refactoring pop-up error when deleting the projects:

```
An exception has been caught while processing the refactoring  
'Delete Resource'
```

If you click Details (in the lower right) you will observe the reason

```
Problems encountered while deleting resources.  
    Could not delete '/DonateJSFWeb/WebContent'.  
    Could not delete 'C:\workspaces\ExperienceJEE\DonateJSFWeb'.
```

This error occurs when a Web session is still active and as result the Faces libraries are locked. You can resolve this error by restarting the test server:

1. At the Refactoring pop-up, click **Abort** to exit the pop-up.
2. In the Servers view, select **ExperienceJEE Test Server**, right-click and select **Restart**.
3. After the server is restarted, delete the remaining projects (and remember to select **Delete project contents on disk**).

## A.2 Chapter specific jump start steps

### Important!



Complete only the following steps that apply for your situation:

- ▶ You do not have to complete a step if you have already successfully completed the referenced chapter.
- ▶ You do not have to complete a step if it is part of a chapter after the one that you are jump starting.

For example, if you are jump starting Chapter 8, “Create the Web front end” on page 277, you only have to complete steps 3, 4, 5, and 6.

Remember the administrative user ID and password used to create the profile in 3.4, “Create the ExperienceJEE server profile” on page 62. You have to provide the user ID and password in the commands.

1. Extract the **SG247827\_JUMPSTART.ZIP** samples file:

-  Unzip to C:\ to create C:\7827code.
-  Unzip to \$HOME/ to create \$HOME/7827code.

Refer to Appendix B, “Additional material” on page 629 for download instructions.

2. For all chapters after Chapter 5, “Create the JPA entities” on page 123, import the starting archive for the chapter into the workspace:
  - a. Ensure that the Rational Application Developer workbench is started (as described in 3.3, “Create the workspace” on page 57).
  - b. If you have any existing projects, remove them from the server and delete them from the workspace as described in A.1, “Common jump start activities” on page 616.
  - c. In the workbench action bar, select **File** → **Import**.
  - d. At the Import pop-up, select **Other** → **Project Interchange** and click **Next**.
  - e. At the Import Project Interchange Contents pop-up:
    - i. Next to From zip file, click **Browse**.

- ii. At the resulting navigation pop-up, navigate to C:\7827code\jumpstart and select the Project Interchange file for the chapter **prior** to the one you are jump-starting and click **Open..**

**Behind the scenes:** the project interchange file names that were captured at the end of each chapter are shown in Table 16-4 on page 619.

Table 16-4

Chapter	Title (abbreviated)	Project Interchange file at end of chapter
5	JPA entities	ch5-jpa.zip
6	Session facade	ch6-facade.zip
7	Session bean	ch7-session.zip
8	Web front-end	ch8-web.zip
9	Application client	ch9-appclient.zip
10	Core security	ch10-security.zip
11	Web service create	ch11-webservice.zip
12	Web service security	ch12-webservice-security.zip
13	MDB create	ch13-mdb-create.zip
14	MDB pub/sub	ch14-mdb-pubsub.zip
15	MDB security	ch15-mdb-security.zip
16	Web 2.0	ch16-web20.zip

- f. Back at the Import Project Interchange pop-up:
  - i. Click **Select All** to enable all projects contained in the project interchange.
  - ii. Click **Finish**, and wait until the workspace is built.

### Off course?

You may observe an error on DonateJSFWeb:

Faces resources need to be updated (use Quick Fix).

This error indicates that one or more of the Faces libraries contained in DonateJSFWeb are downlevel from the versions available in the workbench.

You can update the libraries and resolve this error by:

1. In the Problems view, highlight the error, right-click and select **Quick Fix**.
2. At the resulting Quick Fix pop-up, click **Finish** to execute the default action of updating the resources in DonateJSFWeb
3. At the resulting Update resources, pop-up, click **Yes** to update the indicated libraries. For example, Figure 16-28 illustrates the situation where the html\_extended library is newer. You may receive a different list depending on the exact version of your workbench.

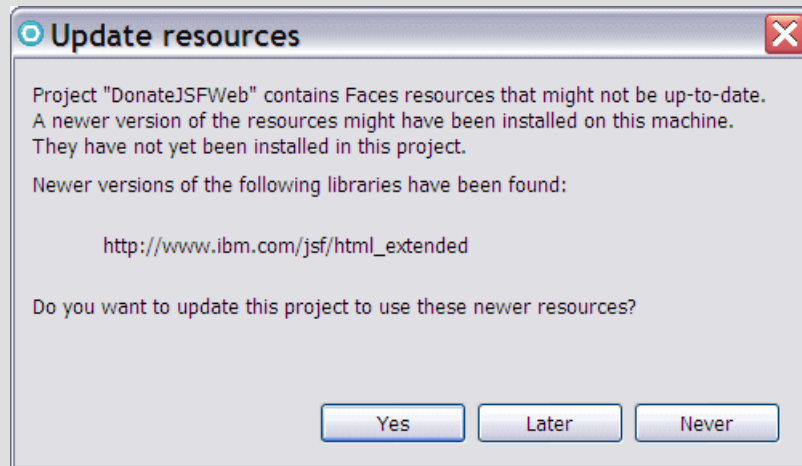


Figure 16-28 Update resources confirmation prompt

4. The errors should be resolved after the library is updated and the workspace is automatically rebuilt.

5. For all chapters after Chapter 5, “Create the JPA entities” on page 123, configure the JPA properties for the DonateJPAEmployee and DonateJPAFund projects:



- a. In the Enterprise Explorer, select **DonateJPAEmployee**, right-click and select **Properties**.
  - i. At the Properties pop-up, select **JPA** (on the left), and set Connection to **Vacation** (on the right).
  - ii. Click **OK**.
- b. In the Enterprise Explorer, select **DonateJPAFund**, right-click and select **Properties**.
  - i. At the Properties pop-up, select **JPA**, and set Connection to **Donate**.
  - ii. Click **OK**.

#### Behind the scenes:

The JPA properties are stored in the workspace .metadata subdirectory and not under the actual project directory. For example, for DonateJPAEmployee the properties are stored in:

```
<workspace>\.metadata\.plugins\org.eclipse.core.resources\proj  
ects
```

```
\DonateJPAEmployee\indexes\properties.index
```

Therefore, they are not included as part of the standard project archive format, and must be reconfigured after importing a project archive.

Note that you get JPA warnings in the Problems view if the connection is not specified or is not active. The application does run anyway, but it is advisable to take care of the warnings.

- c. If you see any errors:
  - i. In the workbench action bar, select **Project** → **Clean**.
  - ii. At the Clean pop-up, select **Clean all projects**.
  - iii. Click **OK**.
- d. The projects exist now in the workspace, with possibly some errors and warnings (see **Errors and warnings** box next). The exact errors and warnings vary depending on which chapter you are jump starting.

### Errors and warnings:

- ▶ If you have no connection to the two databases, you get warnings:

No connection specified for project. No database-specific validation will be performed.

See step 5 on page 620 for a solution.

- ▶ If you have the connections defined, but you have not previously executed the steps in 5.8.4, “Run the JUnit test” on page 186 (for testing the Fund entity), three errors will be displayed indicating that the column and schema information cannot be found for the FUND table:

Column "balance" cannot be resolved

Column "name" cannot be resolved

Schema "DONATE" cannot be resolved for table "FUND"

The errors exist at this point because the FUND table has not yet been created and configured. We will do this in the next step.

- ▶ Warnings of the nature Type safety: Unchecked cast from List to List<Fund>, can be eliminated by adding the annotation **@SuppressWarnings("unchecked")** before the method with the error.

This annotation can be added using the Eclipse quick fix capability as used in d., “Switch back to the Problems view and verify that the three errors related to DonateJPACFund are no longer displayed.

Note that you still have the error related to DonateEAR (and that error will be resolved in Chapter 6, “Create the entity facade session beans” on page 201).” on page 191.

- ▶ Warnings of the nature The serializable class Xxx does not declare a static final serialVersionUID field of type long, can be eliminated by adding the annotation **@SuppressWarnings("serial")** before the class.

- ▶ Warnings of the nature: No grammar constraints (DTD or XML schema) detected for the document ... can be disabled globally (for all schema files that lack grammar constraints):
  - In the workbench action bar, select **Window** → **Preferences**.
  - At the Preferences pop-up:
    - On the left, select **XML** → **XML Files**.
    - On the right (in the resulting XML Files pane), change the Validating files/Indicate when no grammar is specified from **Warning** to **Ignore**.
    - Click **OK**.
  - The warnings should be eliminated after the next build/clean.
- ▶ Project 'DonateUnitTester' is missing required library: 'C:\IBM\SDP\runtimes\base\_v7\runtimes\com.ibm.ws.admin.client\_7.0.0.jar'
- ▶ Project 'DonateUnitTester' is missing required library: 'C:\IBM\SDP\runtimes\base\_v7\runtimes\com.ibm.ws.jpa.thinclient\_7.0.0.jar'

These errors results if the Rational Application Developer installation on your system is different than on the system where this project interchange was created, and ares resolved by editing the project classpath:

- In the Enterprise Explorer, select **DonateUnitTester**, right-click and select **Properties**.
  - At the resulting pop-up, on the left select **Java Build Path**, and on the right select the **Libraries** tab.
  - On the right, highlight com.ibm.ws.admin.client\_7.0.0.jar and click **Edit**.
  - Navigate to the correct location for com.ibm.ws.admin.client\_7.0.0.jar and click **Open** (**OK** on Linux).
  - Repeat the preceding steps for com.ibm.ws.jpa.thinclient\_7.0.0.jar.
- Click **OK** to save and close the Properties.

6. For all chapters after Chapter 5, “Create the JPA entities” on page 123, run the EntityTester JUnit test to create the Donate database and the FUND table:

- a. In the Enterprise Explorer, select **EntityTester** (in DonateUnitTester/src/donate.test), right-click and select **Run As** → **JUnit Test**.
- b. In the JUnit view, verify that the test completed successfully (0 failures and 0 errors).

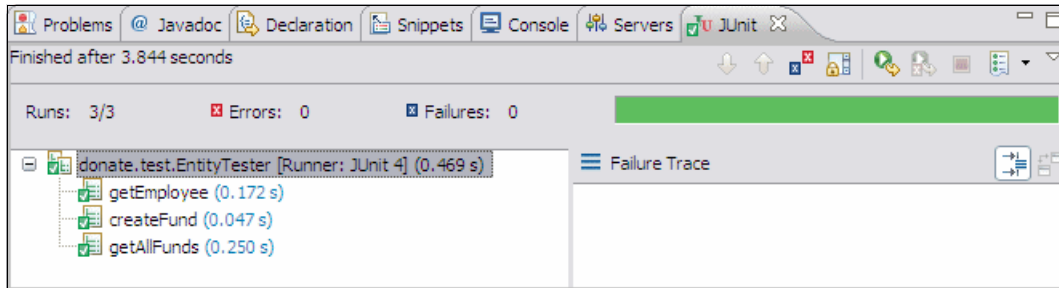


Figure 16-29 JUnit test to create the database and tables

#### Behind the scenes:

The JUnit test fails with a `NullPointerException` if you did not correctly add the user libraries:

- Review 5.8.1, “Create the JUnit project” on page 180 to ensure that the user libraries are properly defined (with the proper JAR files) in the DonateUnitTester properties.

#### Extra credit:

The FUND table can now be defined in the Donate database, and the three errors referencing the missing columns and table should be resolved, if you refreshed the Donate connection and then validated DonateJPAFund (or cleaned the workspace).

7. For all chapters after Chapter 6, “Create the entity facade session beans” on page 201, configure and execute the build.xml script as described in 6.7, “Create the EJB 3.0 stub classes” on page 248.

If the Rational Application Developer installation on your system is different than on the system where this project interchange was created, then you must change the executable path parameter in 6.7.1, “Create the build.xml ANT script” on page 248 to match your installation directory.

8. For all chapters after Chapter 10, “Implement core security” on page 361:

- a. Configure the users and groups as described in 10.3, “Preliminary security setup” on page 365.
- b. Enable Java EE application security as described in 10.4, “Enable Java EE application security” on page 370.

If the Rational Application Developer installation on your system is different than on the system where this project interchange was created, then you must change the `propDir` variable in `DonateBeanTester.java` (in `DonateUnitTester/src/donate.test`).

#### Passwords:

- For all chapters after Chapter 10, “Implement core security” on page 361, update the passwords in `DonateBeanTester`.

Edit **DonateBeanTester** (in `DonateUnitTester/src/donate.test`) and change the “xxxxxxx” password value in the `init` method to match the password used previously:

```
LoginContext lc = new LoginContext("WSLogin",  
    new WSCallbackHandlerImpl("DNARMSTRONG", "xxxxxxx"));
```

- For all chapters after Chapter 13, “Create the message-driven bean” on page 477, update the password used to create the connection in `DonateMDBClient`.

Edit **Main** (in `DonateMDBClient/appClientModule`) and change the password value to match the password used previously.

```
Connection connection = cf.createConnection("javaeeadmin",  
    "xxxxxxx");
```

- For all chapters after Chapter 14, “Add publication of results” on page 517, update the password used to create the connection in `DonatePubSubClient`.

Edit **Main** (in `DonatePubSubClient/appClientModule`) and change the password value to match the password used previously.

```
Connection connection = cf.createConnection("javaeeadmin",  
    "xxxxxxx");
```

- For all chapters after Chapter 15, “Implement security for messaging” on page 535, update the passwords in `DonateMDBClient`:

Edit **Main** (in `DonateMDBClient/appClientModule`) and change the password value to match the password used previously.

```
/* Added for MDB Security  
toutMessage.setStringProperty("userid", "DNARMSTRONG");  
toutMessage.setStringProperty("password", "password");
```

9. For all chapters after Chapter 13, “Create the message-driven bean” on page 477, configure the necessary JMS resources in the ExperienceJEE Test Server as described in 13.4, “Configure the test server for base JMS resources” on page 484.
10. For all chapters after Chapter 5, “Create the JPA entities” on page 123, publish the DonateEAR application.
  - a. Ensure that **ExperienceJEE Test Server** is started as described in 3.6, “Start the ExperienceJEE Test Server” on page 78.
  - b. Ensure that the Derby Network Server is started as described in 4.2, “Start the Derby Network Server” on page 87.
  - c. In the Servers view, select **ExperienceJEE Test Server**, right-click and select **Add and Remove Projects**.
    - i. At the Add and Remove Projects pop-up, select **DonateEAR** in the left pane, and click **Add** to move it to the right pane.
    - ii. Click **Finish**.
  - d. Back in the Servers view, select **ExperienceJEE Test Server**, right-click and select **Publish**.

The deployment takes place in the background, with the status displayed in the lower right task bar. Wait for the task to complete.
11. For all chapters after Chapter 6, “Create the entity facade session beans” on page 201, create the DonationFund record in the Donate database.
  - a. In the Enterprise Explorer, select **FundFacadeTester** (in DonateUnitTester/src/donate.test), right-click and select **Run As → JUnit Test**.
  - b. In the JUnit view, verify that the test completed successfully (0 failures and 0 errors).

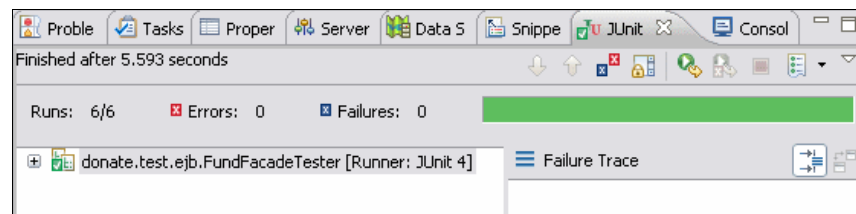
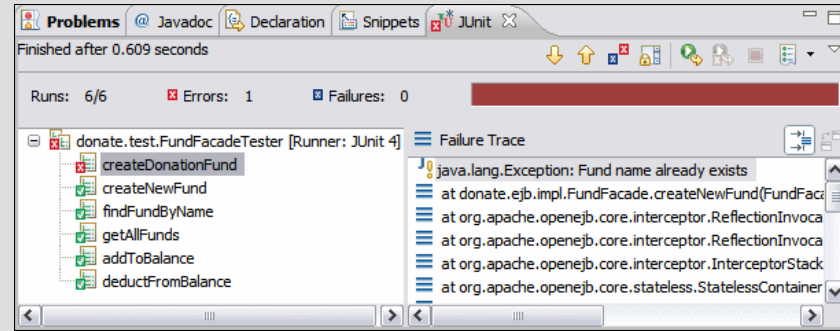


Figure 16-30 JUnit test results

## Off course?

The JUnit test fails with a fund already exists exception if the FundFacadeTester has been run before:



You can ignore this error because the DonationFund can only be created once. However, only the createDonationFund test should fail.

12. For all chapters after Chapter 7, “Create the Donate session bean for the business logic” on page 261, verify the Donate application.
  - a. In the Enterprise Explorer, select **DonateBeanTester** (in DonateUnitTester/src/donate.test), right-click and select **Run As → JUnit Test**.
  - b. In the JUnit view, verify that the test completed successfully (0 failures and 0 errors).

## Behind the scenes:

Executing this JUnit test verifies the donateToFund method in the DonateBean session EJB. This is the core business logic element that is accessed in the various interfaces used in this book: Web, client, Web services, and JMS.

Congratulations! The jump start has been successfully completed.







# B

## **Additional material**

This Appendix provides additional material that you can download from the Internet as follows.

## Locating the Web material

The Web material associated with this IBM Redbooks publication is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG247827>

Alternatively, you can visit the IBM Redbooks Web site at:

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

After you access the Web site, select **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247827.

## Using the Web material

The additional Web material that accompanies this publication includes the following files:

<i>File name</i>	<i>Description</i>
SG247827_BASE.ZIP	ZIP file with base sample code
SG247827_JUMPSTART.ZIP	ZIP file with jumpstart sample code

## System requirements for downloading the Web material

The following system configuration is recommended:

**Hard disk space:** 40 GB minimum

**Operating System:** Windows 2000, Windows XP, Linux

**Processor:** 1.5 GHz or higher

**Memory:** 1 GB or better

Refer to 2.2, “Hardware requirements” on page 26 and 2.3, “Software requirements” on page 27 for further information.

## How to use the Web material

Download the sample code and extract the ZIP file into a suitable directory. Refer to 3.2, “Extract the base sample code file” on page 56 for further instructions.

Instructions on how to use the jump start ZIP files to start at any chapter in the book are provided in Appendix A, “Jump start” on page 615.

# Abbreviations and acronyms

<b>AJAX</b>	Asynchronous JavaScript Technology and XML	<b>JCA</b>	Java Connector Architecture
<b>API</b>	application programming interface	<b>JCP</b>	Java Community Process
<b>AST</b>	Application Server Toolkit	<b>JDBC</b>	Java Database Connectivity
<b>BMP</b>	bean managed persistence	<b>JDK</b>	Java Developer's Kit
<b>CMP</b>	container managed persistence	<b>JMS</b>	Java Message Service
<b>DOM</b>	Document Object Mode	<b>JNDI</b>	Java Naming and Directory Interface
<b>EGL</b>	Enterprise Generation Language	<b>JSF</b>	JavaServer Faces
<b>EJB</b>	Enterprise JavaBeans	<b>JSON</b>	JavaScript Object Notation
<b>ESB</b>	Enterprise Service Bus	<b>JSP</b>	JavaServer Pages
<b>FAQ</b>	frequently asked questions	<b>JSR</b>	Java Specification Request
<b>FTP</b>	File Transfer Protocol	<b>JVM</b>	Java Virtual Machine
<b>GUI</b>	graphical user interface	<b>LDAP</b>	Lightweight Directory Access Protocol
<b>HTML</b>	Hypertext Markup Language	<b>LTPA</b>	Lightweight Third Party Authentication
<b>HTTP</b>	Hypertext Transfer Protocol	<b>MDB</b>	message-driven bean
<b>IBM</b>	International Business Machines Corporation	<b>MQ</b>	message queue
<b>IDE</b>	integrated development environment	<b>MTOM</b>	Message Transmission Optimization Mechanism
<b>ITSO</b>	International Technical Support Organization	<b>MVC</b>	model view controller
<b>J2EE</b>	Java 2, Enterprise Edition	<b>OASIS</b>	Organization for the Advancement of Structure
<b>J2SE</b>	Java 2, Standard Edition	<b>RDB</b>	relational database
<b>JAAS</b>	Java Authentication and Authorization Service	<b>REST</b>	Representational State Transfer
<b>JACC</b>	Java Authorization Contract for Containers	<b>RHEL</b>	Red Hat Enterprise Linux
<b>JAX-RPC</b>	Java API for XML-based RPC	<b>RMI</b>	Remote Method Invocation
<b>JAX-WS</b>	Java API for XML Web Services	<b>RPC</b>	remote procedure call
<b>JAXB</b>	Java Architecture for XML Binding	<b>SAAJ</b>	SOAP with Attachments API for Java
		<b>SAML</b>	Security Assertion Markup Language

<b>SCA</b>	Service Component Architecture	<b>XSD</b>	XML Schema Definition
<b>SDO</b>	Service Data Objects	<b>XSL</b>	Extensible Stylesheet Language
<b>SIB</b>	service integration bus		
<b>SIP</b>	Session Initiation Protocol		
<b>SOA</b>	service oriented architecture		
<b>SOAP</b>	Simple Object Access Protocol (also known as Service Oriented Architecture Protocol)		
<b>SQL</b>	structured query language		
<b>SSL</b>	Secure Sockets Layer		
<b>SVG</b>	Salable Vector Graphics		
<b>SWAM</b>	Simple WebSphere Authentication Mechanism		
<b>SWT</b>	Standard Widget Toolkit		
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol		
<b>UDDI</b>	Universal Description, Discovery, and Integration		
<b>UML</b>	Unified Modeling Language		
<b>URI</b>	uniform resource identifier		
<b>URL</b>	uniform resource locator		
<b>UTC</b>	Universal Test Client		
<b>VM</b>	virtual machine		
<b>VML</b>	Vector Markup Language		
<b>WS-BPEL</b>	Web Services Business Process Execution Language		
<b>WS-I</b>	Web Services Interoperability		
<b>WS-RM</b>	Web Services Reliable Messaging		
<b>WSDL</b>	Web Services Description Language		
<b>WSIL</b>	Web Services Inspection Language		
<b>WTP</b>	Web Tools Platform		
<b>WYSIWYG</b>	what you see is what you get		
<b>XML</b>	Extensible Markup Language		

# 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 publications” on page 635. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297
- ▶ *Rational Application Developer V7.5 Programming Guide*, SG24-7672
- ▶ *Rational Application Developer V7 Programming Guide*, SG24-7501
- ▶ *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611
- ▶ *Web Services Feature Pack for WebSphere Application Server V6.1*, SG24-7618
- ▶ *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257
- ▶ *Building Dynamic Ajax Applications Using WebSphere Feature Pack for Web 2.0*, SG24-7635
- ▶ *Building SOA Solutions Using the Rational SDP*, SG24-7356
- ▶ *WebSphere Application Server V6.1: Planning and Design*, SG24-7305
- ▶ *WebSphere Application Server V6 Security Handbook*, SG24-6316
- ▶ *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304
- ▶ *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212
- ▶ *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135

## Online resources

The following Web sites are a summary of those listed previously in the Learn! and Explore! sections of this publication:

- ▶ WebSphere and Rational software:  
<http://www.ibm.com/software/websphere>  
<http://www.ibm.com/software/rational>
- ▶ WebSphere Information Center:  
<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp>
- ▶ IBM Education Assistant:  
<http://www.ibm.com/software/info/education/assistant>
- ▶ developerWorks:  
<http://www.ibm.com/developerworks>
- ▶ alphaWorks:  
<http://www.ibm.com/alphaworks>
- ▶ Eclipse:  
<http://www.eclipse.org>
- ▶ Web Tools Platform:  
<http://www.eclipse.org/webtools>
- ▶ Eclipse marketplace:  
<http://marketplace.eclipse.org/>
- ▶ Sun Java:  
<http://java.sun.com>
- ▶ Java Community Process:  
<http://www.jcp.org>
- ▶ Apache Derby database:  
<http://db.apache.org/derby>
- ▶ Dojo toolkit:  
<http://dojotoolkit.org/>
- ▶ OASIS:  
<http://www.oasis-open.org>
- ▶ mozilla.org:  
<http://www.mozilla.org>

- ▶ Jython:  
<http://www.jython.org>
- ▶ Web Services Interoperability Organization:  
<http://www.ws-i.org>

## How to get IBM Redbooks publications

You can search for, view, or download Redbooks publications, Redpapers, Hints and Tips, draft publications, and Additional materials, as well as order hardcopies of Redbooks publications or CD-ROMs, at the following 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)







**Redbooks**

# Experience Java EE! Using Rational Application Developer V7.5

(1.0" spine)  
0.875" <-> 1.498"  
460 <-> 788 pages







# Experience Java EE!

## Using Rational Application Developer V7.5

**On-ramp to Java EE development**

**Including EJB 3.0, Web services, JMS and Web 2.0**

**Detailed hands-on tutorials**

This IBM® Redbooks® publication is a hands-on guide to developing a comprehensive Java™ EE application using Rational® Application Developer V7.5 and the embedded WebSphere® Application Server V7.0 test environment, including items such as core functions, security, Web services, and messaging.

Novice users are thus able to experience Java EE, and advance from theoretical knowledge gained by reading introductory material to practical knowledge gained by implementing a real-life application.

Experience is one stage in gaining and applying knowledge, but there are additional stages needed to complete the knowledge acquisition cycle. This book also helps in those stages:

- ▶ Before experiencing Java EE, you learn about the base specifications and intellectual knowledge of Java EE through brief descriptions of the theory and through links to other information sources.
- ▶ After experiencing Java EE, you explore advanced Java EE through previews of advanced topics and links to other information sources.

The target audience is technical users who have minimal experience with Java EE and the IBM WebSphere Application Server product set, but who do have past experience in using an integrated development environment (IDE) and in architecting or designing enterprise applications.

**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)