WebSphere. software

IBM

Experience J2EE! Using WebSphere Application Server V6.1

Including Web services and messaging

Using the Application Server Toolkit

Using the Derby database

Charles P. Brown

Redbooks

ibm.com/redbooks

International Technical Support Organization

Experience J2EE! Using WebSphere Application Server V6.1

January 2007

SG24-7297-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xi.

First Edition (January 2007)

This edition applies to Version 6.1 of WebSphere Application Server and the Application Server Toolkit.

Contents

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which 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

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

AIX®
alphaWorks®
Cloudscape™
CICS®
DataPower®
DB2®
developerWorks®
Informix®
IBM®
iSeries™
i5/OS®
OS/400®
pSeries®
Rational®
Redbooks™
Redbooks (logo) ™
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

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

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

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, JavaBeans, JavaMail, JavaScript, JavaServer, JavaServer Pages, JDBC, JDK, JMX, JSP, JVM, J2EE, J2SE, Solaris, Sun, Sun Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Itanium, 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, other countries, or both.

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

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

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

Preface

This IBM® Redbook is a hands-on guide to developing a comprehensive J2EE™ application, including core functions, security, Web services, and messaging. In the context of J2EE, messaging is also known as Java™ Message Service (JMS).

Novice users are thus able to experience J2EE, 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 IBM Redbook also helps in those stages:

- Before experiencing J2EE, you *learn* about the base specifications and intellectual knowledge of J2EE through brief descriptions of the theory and through links to other information sources.
- After experiencing J2EE you will *explore* advanced J2EE through previews of advanced topics and links to other information sources.

This publication is not intended to duplicate the numerous excellent documents, tutorials, and demonstrations that provide both basic J2EE introductions (Learn) and expert level information (Explore) on specific J2EE aspects. Many of the other resources are available at the following Web sites:

- IBM developerWorks®

  http://www.ibm.com/developerworks
- IBM Redbooks™

  http://www.redbooks.ibm.com
- Sun™ J2EE (aka Java EE)

  http://java.sun.com/javaee

The target audience is technical users who have minimal experience with J2EE and the IBM J2EE product set, but who do have past experience in using an integrated development environment (IDE) and in architecting or designing enterprise applications.

The individual that wrote this redbook

Charles P. Brown (Charlie Brown), an IBM Senior Consulting IT Specialist in the WebSphere® brand of products, wrote this IBM Redbook. He spent over 20 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).

This book is the result of contributions by many individuals, and the author would like to thank the following for their contributions:

- Jim Ryan, Consulting IT Architect, IBM (Poughkeepsie, NY, USA), whose frustration with disjointed J2EE enablement material provided the initial inspiration for this publication.
- Thierry Matusiak, IT Specialist, IBM (Courbevoie, France), for his role as a contributing author, providing updated material, feedback on document content and format, updated graphics, and suggestions on how to improve overall usability and readability.
- Ueli Wahli, Consulting IT Specialist. IBM (San Jose, CA, USA) for his role as the IBM Redbook project leader, providing the guidance needed to produce this publication.
- Wini Mark, Staff Software Engineer, IBM (Toronto, Canada) for providing assistance on the migration from Rational® Application Developer V6.0 to WebSphere Application Server Toolkit V6.1.
- William Zobrist, formally of IBM Press, for his critical review of this material and for challenging me to refine the overall theme of my draft book, leading to the Learn, Experience, Explore organization used in this book.
- My manager, Kent Keeling, IBM (Broomfield, CO, USA), for providing a work environment that encourages activities like writing a book.
- John Dale, Jaylani Sharif, and Anders Vinther (all from IBM) for reviewing the draft version of this book.

And thanks to the many others who provided feedback on the previous IBM internal version of this book (called the "Whirlwind Tour of J2EE").

Become a published author

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

Your efforts will help increase product acceptance and client satisfaction. As a bonus, you will 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 the following Web site:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@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, you perform the following:

- Read an introduction to both J2EE and the scenario used in this redbook
- Install and configure the pre-requisite software
- Configure the development environment
- Create and populate the database that represents the legacy application

1

Introduction

In this chapter we introduce you to the following items:

- The purpose and intent of this IBM Redbook
- An overall view of J2EE
- 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 J2EE. Is there a good introductory hands-on book—not J2EE for Dummies but something that provides a cohesive view of J2EE with enough information to allow them to develop a real-life application?*
>
> *Is there a book that helps them to experience J2EE?”*

I heard this same basic question on many technologies during my 20-plus career in the Information Technology (IT) industry, most recently with Java 2 Platform, Enterprise Edition (J2EE). Refer to 1.2 “J2EE simplified” on page 7 for a brief introduction to J2EE.

J2EE developers have access to the many Web sites, books, articles, and newsgroups that deal with J2EE and related technologies. Still, new J2EE developers find it very difficult to move from reading this mass of information to actually applying the technology to solve business issues.

The British have a saying that refers to something being “used in anger.” The usage is more than just reading or playing: Something is actually done. In this, case, how do developers effectively “use J2EE in anger”? What is an optimal way to obtain this knowledge, and what are they not finding?

Over time, I have defined the following three phases of acquiring and applying knowledge of a technology:

- 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.

My experience suggests that generally available information on 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.

Take the following for example:

- 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 may understand that subject area but does not understand how it links with security for Web front ends, thick clients, Web services, and messaging (JMS). This is the adage of "not being able to see the forest for the trees."
- The reader must perform repetitive basic configuration tasks in each scenario (since each guided tutorial/example uses a separate sample application). This increases the amount of time he/she 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 J2EE aspect (including the proverbial kitchen sink). This works well for experienced J2EE developers who use the material as a reference, but it can overwhelm and confuse the J2EE novice.

Experience UNIX®?

This problem is not unique to J2EE. 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?

My contention is that 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 like awk, sed, and grep. The unusability was adopted as a badge of honor and defined the technology and the community. "Novice users need not apply!"

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. Better late than never!

In 2003, I worked with a team of individuals trying to provide initial J2EE training using a series of lectures and labs that came from multiple independent sources. We were constantly faced with the problems I described in the Experience UNIX note box. One of my co-workers expressed frustration about this and said that if we had time, we should create a set of material that uses a single-integrated scenario to address these problems.

This book is the result, enabling a reader to *experience* the full spectrum of basic J2EE application development and deployment in a single document (that is not 2000 pages in length):

- This publication utilizes a single-integrated scenario across all of J2EE. You learn and set up a single application, and sequentially add the various J2EE 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 of the scenario” on page 19 describes the scenario used in this book. The following IBM software products will be used to implement the J2EE application:

J2EE runtime:	WebSphere Application Server V6.1 server
J2EE development tool:	WebSphere Application Server Toolkit V6.1 (AST V6.1)

- I keep the book 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 sites referenced in this book:

  – IBM developerWorks

    `http://www.ibm.com/developerworks`

  – IBM Redbooks

    `http://www.redbooks.ibm.com`

  – Sun J2EE (aka Java EE)

    `http://java.sun.com/javaee`

  – IBM Education Assistant

    `http://www.ibm.com/software/info/education/assistant`

The target readers are technical users who have minimal experience with the J2EE 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 J2EE expert by reading and experiencing this book, but afterwards you should have enough knowledge for the following:

- Develop, test, and deploy simple J2EE applications that contain necessary qualities of service such as 2-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 will help you expand your knowledge and apply the appropriate technology to business scenarios.

Use this IBM Redbook as a self-guided tutorial, as demonstration scripts, or as hands-on labs for a workshop.

I hope this book leads you to a successful J2EE experience!

1.2 J2EE simplified

"The Java 2 Platform, Enterprise Edition (J2EE) is a set of coordinated specifications and practices that together enable solutions for developing, deploying, and managing multi-tier server-centric applications. Building on the Java 2 Platform, Standard Edition (J2SE), and the J2EE platform adds the capabilities necessary to provide a complete, stable, secure, and fast Java platform to the enterprise level. It provides value by significantly reducing the cost and complexity of developing and deploying multi-tier solutions, resulting in services that can be rapidly deployed and easily enhanced."

From the J2EE Frequently Asked Questions (FAQs), at the following Web site:

http://java.sun.com/javaee/overview/faq/j2ee.jsp

What? I understand J2EE, and I find this definition confusing! What is J2EE and why do you care?

The Sun Java™ site contains a good introduction that answers these questions. The introduction is called the **Simplified Guide to the Java 2 Platform, Enterprise Edition** and is available at the following Web site:

http://java.sun.com/j2ee/reference/whitepapers/j2ee_guide.pdf

The Sun Java site also contains a comprehensive tutorial called **The J2EE 1.4 Tutorial,** located at the following Web site:

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

If you find these to be confusing as well, following is my simplified description of J2EE. The key point to remember is that J2EE allows developers to create applications using COTS (commercial off the shelf) development software, and then provides these applications to others to install and execute on COTS runtime software.

1.2.1 Objective

> The J2EE 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).

J2EE acts as the mediator between the application, the operating system, and the back-end services. J2EE masks the differences among the various operating systems and back-end services, providing a consistent set of interfaces and functions to the application. This allows a single version of the source code to support all environments. The term that Sun uses is WORA (Write Once, Run Anywhere).

J2EE, 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:

- J2EE runtime providers (IBM, BEA, Oracle®, Sun)
- Operating systems (Microsoft Windows, Unix, Linux, AIX®)
- Databases (DB2®, Oracle Database, Sybase, Microsoft SQL Server)
- Back-end information systems (SAP®, PeopleSoft®, and others)

In practice, J2EE 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:

- J2EE 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.
- J2EE 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 J2EE runtime providers.
- The J2EE runtime cannot completely mask the differences between the various environments. For example, DB2 and Oracle Database have different requirements for table names.

I am not denigrating J2EE by asserting that it cannot support 100% code portability in all situations. Rather, I am trying to set realistic expectations.

My experiences suggest that most J2EE applications exceed 99% code portability—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

J2EE 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.

J2EE 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 J2EE 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 J2EE platform standard.
- Reference implementation that provides a rudimentary implementation of the J2EE specifications. This 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 J2EE standards

The J2EE standards were first formally released in 1999 and continue to evolve.

J2EE version	Based on J2SE	Final specification release date
1.2	1.3	12/1999
1.3	1.3	09/2001
1.4	1.4	11/2003
1.5 (5.0)	1.5 (5.0)	05/2006

Note that the naming structure changed with the latest version from Java 2 Platform, Enterprise Edition (J2EE) 1.x to Java Platform, Enterprise Environment (Java EE) x. For example, the v1.5 specification is formally called Java EE 5 instead of J2EE 1.5. However, since this book focuses on v1.4, I will stick with the J2EE 1.x convention.

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

The specification updates are also changing in focus. Earlier J2EE releases added new functions and features, but many of the changes in J2EE 1.5 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 annotation 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 J2EE platform. The platform provides enough base functions for users to develop highly-functional and portable applications. Users are generally satisfied with the current functions, and now are looking instead at how it can be done better and simpler.

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

http://jcp.org

JCP works on a broad range of specifications called Java Specification Requests (JSRs). These JSRs may or may not be incorporated into new versions of J2EE and J2SE, based on input from the open community.

The J2EE 1.4 specification is formally called *JSR 151: JavaTM 2 Platform, Enterprise Edition 1.4 (J2EE 1.4) Specification*, by Sun, and is available at the following Web site:

```
http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf
```

The J2EE 1.5 specification is formally called *JSR 244: JavaTM 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
```

1.2.3 Implementations

Software companies use these specifications to create and sell development and runtime J2EE software products.

J2EE is owned by Sun and licensed to companies that must pass the included compatibility test suite before shipping a product with the J2EE logo.

For a list of current authorized licensees of the J2EE Platform, visit the following Web site:

```
http://java.sun.com/j2ee/licensees.html
```

These software products are often termed COTS (commercial off-the shelf) software.

The following table provides some key J2EE product providers.

Vendor	Runtime product	IDE product
IBM	WebSphere Application Server	Rational Application Developer, Application Server Toolkit
	WebSphere Community Edition	Eclipse plug-ins
BEA	WebLogic Server	WebLogic Workshop
Oracle	Oracle Application Server	JDeveloper
Sun	Sun Java System Application Server	Sun Java Studio Enterprise

Vendor	Runtime product	IDE product
JBoss	JBoss Application Server	JBoss Eclipse IDE

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 runtimes products due to the following:

- Some of the minor variations in J2EE 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 J2EE.
- Testing and debugging tools for that particular runtime product.

Behind the scenes:

IBM provides two different sets of tooling and runtime suites. This redbook uses the WebSphere Application Server (V6.1) runtime and the WebSphere Application Server Toolkit (V6.1) tooling.

- **WebSphere Application Server** is the IBM premier J2EE 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).

  Other infrastructure capabilities provided by IBM are based on WebSphere Application Server:

  - Portal (WebSphere Portal Server)
  - Enterprise Service Bus (ESB) (WebSphere Enterprise Service Bus)
  - WS-BPEL based workflow (WebSphere Process Server)
  - e-commerce (WebSphere Commerce)

  IBM has two different versions of the WebSphere Application Server that implement the J2EE 1.4 specification:

  - The WebSphere Application Server V6.0 runtime uses the J2SE 1.4 Java environment.

    J2EE applications for WebSphere Application Server V6.0 are developed and tested using Rational Application Developer V6.0 and V7.0.

  - The WebSphere Application Server V6.1 runtime uses the Java 5 (J2SE 1.5) environment.

    J2EE applications for WebSphere Application Server V6.0 are developed and tested using Rational Application Developer V7.0 and the WebSphere Application Server Toolkit (AST) V6.1.

    Rational Application Developer V6.0 cannot be used to develop and test applications on Application Server V6.1 because it does not support Java 5.

  AST is a feature of the WebSphere Application Server (both V6.0 and V6.1) but only AST V6.1 version can be used to develop and test J2EE application. AST V6.0 can only be used to modify the configuration of J2EE applications that were built using another IDE.

  Refer to 3.1 “Learn!” on page 42 for more information on Rational Application Developer.

- **WebSphere Community Edition** is a “free lightweight Java 2 Platform, Enterprise Edition (J2EE) application server built on Apache Geronimo...” per the following IBM Community Edition Web site:

  `http://www.ibm.com/software/webservers/appserv/community`

  Free means that the product has a zero cost purchase, but support is fee-based.

  Community Edition is intended for small single-server implementations that require the programming model specifications but not the additional infrastructure support provided by products such as WebSphere Application Server.

  Community Edition ships with Eclipse plug-ins that in conjunction with other packages (Eclipse, Web Tools Platform, and several others) provide an integrated development environment. Refer to the on-line documentation for further information on installing, configuring, and using the Community Edition tooling:

  `http://publib.boulder.ibm.com/wasce/V1.0.0.1/Tasks/Eclipse/index.html`

1.2.4 Web services and messaging

J2EE provides both traditional Internet server and program-to-program interaction through Web services and messaging.

J2EE 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, J2EE 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 303) and messaging (discussed in “Part 4 Messaging” on page 347).

1.2.5 Container architecture

The J2EE 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 J2EE container manages the common services such as security, session state, transactions, database access, fail-over, and recovery.

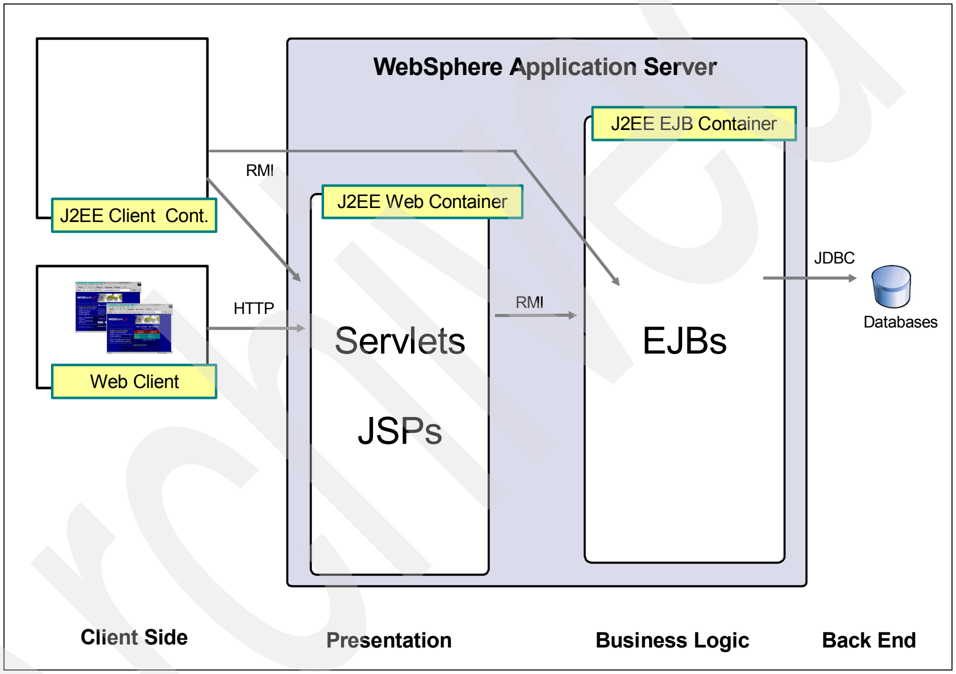


Figure 1-1 J2EE container architecture

The following description of Figure 1-1 is meant to provide a brief introduction to the overall J2EE container environment and programming artifacts. The individual scenario chapters will provide more detail on the specific J2EE components (for example Servlets, JSPs, EJBs).

J2EE 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 from the left in Figure 1-1.

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. The Web browser may support the execution of applets or client-side scripting (such as JavaScript™), but these are not defined by J2EE and do not have access to J2EE services.

  This is a thin client: The user does not have to install any software on the client system (beyond the Web browser).

- The J2EE Client container supports the execution of Java programs (for example, a Java class file with a main method) that have access to J2EE services such as security, transactions, naming, and database access.

  This is a thick client: The user must install a J2EE 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 J2EE Client container; however, the end result is the same: The J2EE Client container is installed on the client system.

(Not shown above)

Many J2EE 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 I do not like to use that name because others use the name Web Tier interchangeably with the Presentation Tier. I recommend calling this the DMZ (“de-militarized zone”) 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 J2EE 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 J2EE application server. Therefore, for large environments, implementing separate HTTP servers can lead to a smaller overall J2EE 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 J2EE 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 may 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 may choose to do this in small environments (where you do not wish to stand up an external HTTP server in addition to the J2EE application server) or when you want to control access to the resources (using J2EE declarative security).

Web containers typically contain a “minimal” HTTP server that directly receives requests from the Web browsers. Therefore, the J2EE 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 J2EE 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 J2EE EJB container provides the services needed to support the access to and execution of business logic. Business logic and data may be accessed concurrently by many different programs on many different systems. The J2EE 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 J2EE implementation of the business logic, whether data (Entity EJBs) or logic (Session EJBs).

In a non-J2EE environment, management of the transactional interactions and local/remote invocation routines can represent a large percentage of the user code. In a J2EE environment, the Business Logic Tier manages this, dramatically reducing the amount of user written code.

Back End Tier

The Back End Tier represents existing business systems, whether databases or other software systems:

- The Java Connector Architecture (JCA) (not shown above) provides a common programmatic way of connecting with non-J2EE 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 of the scenario

You are probably still confused after reading the Introduction of this IBM Redbook and the "**Simplified Guide to the Java 2 Platform, Enterprise Edition**" located on the Sun Web site. A novice can still reasonably ask: How do I apply J2EE to a business need?

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 J2EE could lead you to ask questions such as the following:

- *Should I access the two back-end databases using database/JBDC?*
- *Should I access them using messaging/JMS?*
- *Should I access them using a Web service?*
- *Should I access them using an application connection like 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 J2EE 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 J2EE?

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

- *The requirement is to integrate an existing Vacation database (containing a table called Employee 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).*

Donate sample application

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

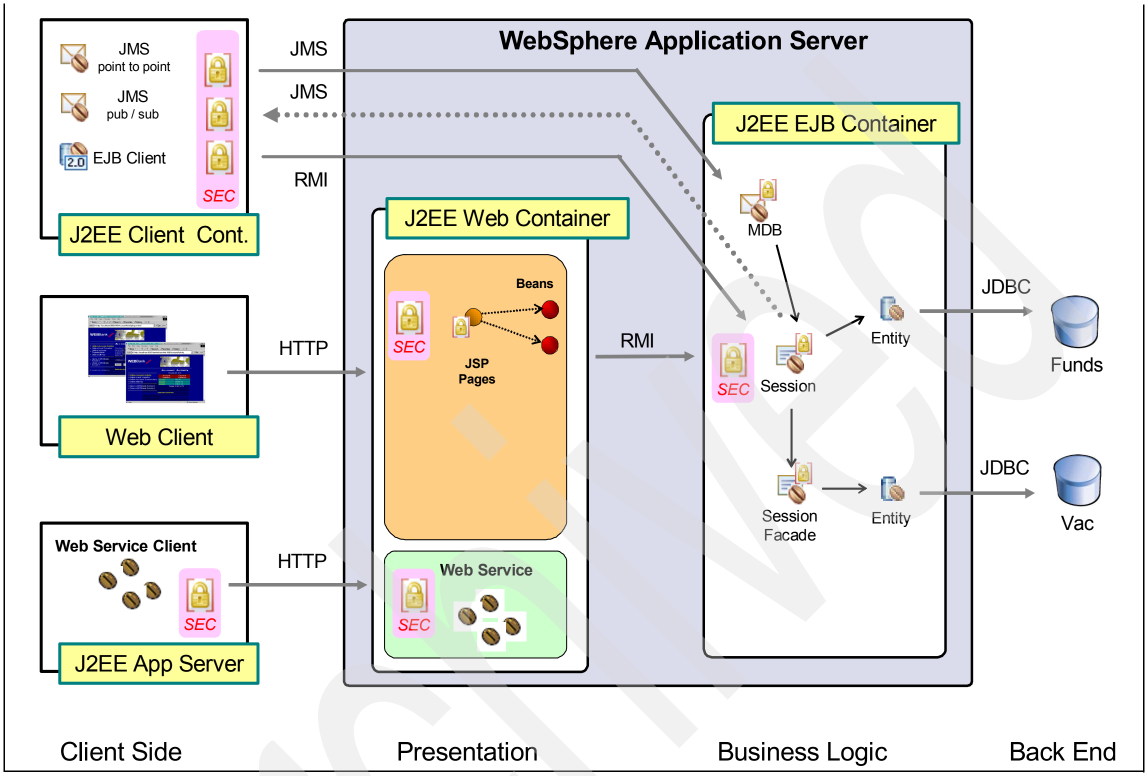


Figure 1-2 Donation sample application scenario

In this book you implement this application by following the following parts and chapters in the following order:

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 23: Ensure that the prereq products are installed.
- Chapter 3, "Configure the development environment" on page 41: Prepare the development workspace and test server.
- Chapter 4, "Prepare the legacy application" on page 71: Extract the application samples needed for this book, and create and populate the Vacation database that represents the current application.

Part 2, "Core J2EE application" on page 81: Create the core J2EE artifacts (data access elements, application business logic elements, Web front end, and thick client):

- Chapter 5, "Create the employee data access element" on page 83: Create the Employee entity EJB (based on the Employee table in the existing Vacation database).
- Chapter 6, "Create the funds data access element" on page 135: Create the Funds entity EJB and the Funds table/database (based on the Funds entity EJB).
- Chapter 7, "Create the donate business logic element" on page 165: Create the Donate session EJB to transfer vacation from an Employee entity EJB to a Funds entity EJB.
- Chapter 8, "Create the employee facade business logic element" on page 195: Create the EmployeeFacade session EJB that provides mediated access to the Employee entity EJB.
- Chapter 9, "Create the Web front end" on page 213: Create a Web application that provides access first to the EmployeeFacade session EJB, and then to the Donate session EJB.
- Chapter 10, "Create the application client" on page 245: Create a thick Java client that invokes the Donate session EJB.
- Chapter 11, "Implement core security" on page 257: Add the needed users to the user registry (in a file system based registry), configure the test server for J2EE 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).

Alternative:

If you already have a strong grasp of core J2EE technologies and your primary interests in this document are the Web services and/or messaging scenarios, you can replace all of the scenarios previously mentioned with:

- Chapter 12, "Alternative Import the core J2EE application" on page 299: Import/configure the completed sample application into AST V6.1.

Part 3, "Web services" on page 303: Extend the core J2EE application to support Web services:

- Chapter 13, "Create the Web service" on page 305: Utilize the AST V6.1 wizards to expose the Donate session EJB as a Web service, and create a Web service client to access this Web service.

- Chapter 14, “Implement security for the Web service” on page 331: 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 347: Extend the core J2EE application to support messaging (JMS):

- Chapter 15, “Create the message-driven bean” on page 349: 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 16, “Add publication of results” on page 387: 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 17, “Implement security for messaging” on page 405: Update the messaging artifacts to support security.

2

Install and configure software

This chapter describes the basic steps to install the software used in this book.

2.1 Required software

The following software packages are used in this book:

- WebSphere Application Server V6.1 server (WAS V6.1), which provides the J2EE runtime.
- WebSphere Application Server Toolkit V6.1 (AST V6.1), which provides the J2EE integrated development environment (IDE).

You can skip this chapter if you already have these products installed. However, this book assumes that you installed to the following directories:

Product	Installation directory used in this book
WebSphere Application Server V6.1	Windows C:\IBM\AppServer Linux /opt/IBM/AppServer
WebSphere Application Server Toolkit V6.1	Windows C:\IBM\AST Linux /opt/IBM/AST

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

2.2 Hardware requirements

The AST V6.1 hardware requirements are listed in the `readme_install_ast.html` file (found in the readme directory on the AST V6.1 installation CD 0), and the WAS V6.1 hardware requirements are listed at the following Web site:

`http://www.ibm.com/software/webservers/appserv/was/requirements`

However, these are generalized requirements and do not reflect the actual installation and usage that drives the true hardware requirements.

The following summarizes the hardware recommendations resulting from the creation of this document:

- **Processor**: We developed the scenarios and instructions in this book on an IBM Thinkpad model T41p that has a Pentium® M 1700 MHz processor, and performance was acceptable.

  The AST V6.1 support information recommends a minimum of a Pentium III 800 mhz processor. The WAS V6.1 support information recommends a minimum of a Pentium 500 MHz processor.

What is the true minimum to complete the scenario and instructions in this book? The AST V6.1 recommendation (Pentium III) is probably a good minimum, but with large software applications, the faster the processor the better.

- **Memory**: Both AST V6.1 and WAS V6.1 recommend a minimum of 500 MB RAM with 1 GB RAM recommended.

  The 1GB RAM recommendation is a good minimum. However, the best value for your environment depends on what else is running on your system. The AST V6.1 process uses upwards of 300+ MB RAM, and the WAS V6.1 process could use upwards of 200+ MB RAM.

  Therefore, your system should be able to generate acceptable performance when the total physical RAM in use is 500 MB higher than what is currently in use on your system (before installing and running AST V6.1 and WAS V6.1).

  For example, if your system is currently using around 300 MB of physical RAM then in theory you should have at least 800 MB RAM. Since Windows performance degrades significantly once memory begins to swap, you would find performance to be marginal with 756 MB RAM and much better with 1 GB RAM.

- **Disk space**: The actual AST V6.1 and WAS V6.1 maximum disk space usage observed during the development of this book under both Microsoft Windows and Linux was approximately 2.7 GB.

Component	Directory (Windows)	Disk space (in MB)
Application Server	C:\IBM\AppServer	1659.0
Update Installer	C:\IBM\UpdateInstaller	877.4
Toolkit	C:\IBM\AST	164.6
Workspace	C:\Workspace\ExperienceJ2EE	49.6
Total		2750.7

Note: Add more disk space if you are installing from product installation images instead of from a product CD-ROM.

2.3 Obtaining the software

The packages used in this book (WAS V6.1 and AST V6.1) are in fact two components of the same product, WebSphere Application Server V6.1. You have access to this product through several channels:

- You may have purchased this product.
- You may be an IBM business partner with the ability to download evaluation copies of the software. Reference **IBM Software Access Catalog** at the following Web site:

  `http://www-304.ibm.com/jct09002c/isv/welcome/softmall.html`

- You may be a professor or student that belongs to the IBM Academic Initiative, and thus have the ability to download evaluation copies of the software. Reference **IBM Academic Initiative** at the following Web site:

  `http://www-304.ibm.com/jct09002c/us/en/university/scholars`

What if none of these channels apply to you? Currently, you cannot follow the scenarios in this IBM Redbook. Although IBM has a 60-day trial version for WebSphere Application Server V6.1, this package only contains the server component (WAS V6.1). The toolkit component (AST V6.1) does not have a trial version.

The Application Server V6.1 trial download is at the following Web site:

`http://www.ibm.com/developerworks/downloads`

2.4 Supported operating systems

The installation instructions and scenarios in this book were created using the following operating systems:

- Microsoft Windows XP Professional Version 2002 (Service Pack 1)
- Red Hat Enterprise Linux (RHEL) WS, Version 4, update 2

Table 2-1 on page 27, based on product support information, lists the supported operating systems on x86 based (Intel®) computers for this combination of products.

Table 2-1 Windows and Linux operating systems (x86)

Operating system	WebSphere Application Server Toolkit V6.1.x	WebSphere Application Server V6.1.x
Microsoft Windows 2000 Advanced Server	Y (SP 4)	Y (SP 4)
Microsoft Windows 2000 Professional	Y (SP 4)	Y (SP 4)
Microsoft Windows 2000 Server	Y (SP 4)	Y (SP 4)
Microsoft Windows Server® 2003, Datacenter		Y (SP 1)
Microsoft Windows Server 2003, Enterprise Edition	Y (SP 1)	Y (SP 1)
Microsoft Windows Server 2003, Standard Edition	Y (SP 1)	Y (SP 1)
Microsoft Windows XP Professional	Y (SP 2)	Y (SP 2)
Red Hat Enterprise Linux AS, Version 3	Y (U 5, 6)	Y (U 5, 6)
Red Hat Enterprise Linux AS, Version 4	Y (U 2)	Y (U 2)
Red Hat Enterprise Linux ES, Version 3	Y (U 5, 6)	Y (U 5, 6)
Red Hat Enterprise Linux ES, Version 4	Y (U 2)	Y (U 2)
Red Hat Enterprise Linux WS, Version 3	Y (U 5, 6)	Y (U 5, 6)
Red Hat Enterprise Linux WS, Version 4	Y (U 2)	Y (U 2)
SUSE Linux Enterprise Server (SLES) 9	Y (SP 2 , 3))	Y (SP 2, 3)

Locate detailed support information for these products at the following Web addresses:

- WebSphere Application Server Toolkit V6.1:

  `readme_install_ast.html` file (found in the readme directory on the AST V6.1 installation CD 0)

- WebSphere Application Server V6.1:

  http://www.ibm.com/software/webservers/appserv/was/requirements

Application Server Toolkit V6.1 is only supported on the x86 based hardware architecture, whereas Application Server V6.1 is supported on a variety of hardware architectures (Table 2-2 on page 28).

Table 2-2 Hardware / operating systems supported by Application Server V6.1

Hardware architecture	Operating systems
IBM iSeries™	i5/OS®, OS/400®, Linux
IBM pSeries®	AIX, Linux
PA-RISC	HP-UX
Intel Itanium®	HP-UX
IBM System Z	Linux on System Z
SPARC	Sun Solaris™
x86	Linux Windows

2.4.1 Windows operating system considerations

Microsoft Windows installation considerations are documented under **Preparing Windows systems for installation** in the online WAS V6.1 information center:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/tins_winsetup.html`

- The scenarios in this book can be performed by any user ID with local administrator authority.
- Your system must have an internet browser (the built-in Internet Explorer® or Mozilla 1.7.5 or later).
- Your system must be able to extract ZIP files.

2.4.2 Linux operating system considerations

Linux installation considerations are documented under **Preparing Linux systems for installation** in the online WAS V6.1 information center:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/tins_linuxsetup.html`

- Any user (root or non-root) can perform the scenarios in this book. However, one step in the installation section (in this chapter) does require root authority. After that step, the non-root user is acceptable.

Behind the scenes:

Prior versions of WebSphere Application Server required that the installation run under the root user, but starting in WAS V6.1 installation can also be run under non-root users.

Prior versions of this redbook (then called the "Whirlwind Tour of J2EE") required that the scenarios be run as root user because J2EE security was enabled using the local operating system user/group registry, and access to that had to be run as the root user (or using SUDO).

This version of the redbook uses a new WAS V6.1 file system based user registry. Access to this registry does not require root access.

- The information center recommends setting the umask to 022 for installing the products. However, for the purpose of this scenario, the default 002 is acceptable.

  The following is from the **Preparing Linux systems for installation** section in the online information center:

  > *"In addition, select a umask that would allow the owner to read/write to the files, and allow others to access them according to the prevailing system policy. For root, a umask of 022 is recommended. For non-root users a umask of 002 or 022 could be used, depending on whether or not the users share the group. To verify the umask setting, issue the following command:*
  >
  > ```
  > umask
  > To set the umask setting to 022, issue the following command:
  > umask 022"
  > ```

- Your system must have the Mozilla Firefox Web browser installed. If needed, Firefox is available for download from the following Web site:

  `http://www.mozilla.org/products/firefox`

Behind the scenes:

The RHEL V4 distribution contains Firefox, but it is not installed by default. To install Firefox, select the **Custom Software packages to be installed** option, and enable the **Applications → Graphical Internet package**.

In general, it is preferable to use the version of Firefox that ships with most major Linux distributions:

- For AST V6.1 and Eclipse to use Firefox as the internal Web browser, Firefox must be compiled with linkable Gecko libraries; otherwise, the internal web browser option is disabled, and any Web pages launched by AST V6.1/Eclipse will appear in external Web browsers.
- The version of Firefox downloaded from www.mozilla.org is not compiled with the gecko library.
- The versions of Firefox shipped with most major Linux distributions are compiled with the gecko library.

Visit the following Web address for more information on this reference and to read the following FAQs:

- Which platforms support the SWT browser?
- What do I need to run the SWT browser inside Eclipse on Linux/GTK or Linux/Motif?

  `http://www.eclipse.org/swt/faq.ph`

- The **Preparing Linux systems for installation** section in the online information center lists the package pre-requisites for various versions of Linux.

  The infocenter indicated that four additional packages needed to be installed for RHEL WS V4. These packages are included in the RHEL WS V4 installation CDs but are not installed as part of the default installation:

  – compat-libstdc++-33-3.2.3-47.3

  – compat-db-4.1.25-9

  – xorg-x11-deprecated-libs-6.8.1 or xorg-x11-deprecated-libs-6.8.2

  – rpm-build-4.3.3-7_nonptl

  The **Installing and verifying Linux packages** section in the online information center provides an overview on the Linux commands you can use to query the list of installed packages and to install additional packages:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/tins_rhel_packages.html

Behind the scenes:

This redbook was developed using RHEL V4 update 2 and three of the four packages were automatically installed through the update. The only package that we manually installed was rpm-build-4.3.3-11_nonptl (which was uplevel from the package stated in the infocenter).

Following are the steps we used:

- RHEL V4 update2 CD ROM 3 was inserted into the CD ROM drive.
- The su root command was issued (and the root password entered at the prompt.
- The CD ROM was mounted:

  ```
  [root@localhost ast]# mount /dev/cdrom /media/cdrom
  mount: block device /dev/cdrom is write-protected, mounting
  read-only
  ```

- The package was installed:

  ```
  [root@localhost ast]# rpm -ivh
  /media/cdrom/RedHat/RPMS/rpm-build-4.3.3-11_nonptl.i386.rpm
  warning:
  /media/cdrom/RedHat/RPMS/rpm-build-4.3.3-11_nonptl.i386.rpm: V3
  DSA signature: NOKEY, key ID db42a60e
  Preparing...
  ########################################### [100%]
     1:rpm-build
  ########################################### [100%]
  ```

- The package installation was verified:

  ```
  [root@localhost ast]# rpm -qa | grep "rpm-build"
  rpm-build-4.3.3-11_nonptl
  ```

- The root user was logged out (by typing exit).

2.5 Install WebSphere Application Server V6.1 (base)

Perform the following steps to install Application Server V6.1 with the configuration used in this book:

1. Linux Log on as the root user, and create a directory under which the WAS V6.1 and AST V6.1 packages can be installed by the non-root user.

   - Run the following command to create the /opt/IBM directory:

     ```
     mkdir /opt/IBM
     ```

   - Run the following command to transfer ownership of this directory to the existing testuser user and the existing testgroup group:

     ```
     chown testuser:testgroup /opt/IBM
     ```

2. Log on as the appropriate user:
   - Windows User belonging to the local administrators group.
   - Linux Either the user specified above in the **chown** command, or a user belonging to the group specified above in the **chown** command.
3. Start the installation program from the WebSphere Application Server image directory (or from the product CD):
   - Windows **.\WAS\install.exe**
   - Linux **./WAS/install**
4. At the **Welcome...** pop-up, click **Next**.
5. At the **Software License Agreement** pop-up, select **I accept both the IBM and non-IBM terms**, and click **Next**.
6. At the **System prerequisites check** pop-up, click **Next**.
7. Linux At the **A Non-root/Non-Administrator user is detected** pop-up, click **Next**.
8. At the **Install Sample Applications** pop-up, select **Install the sample applications**, and click **Next**.
9. At the **Installation directory** pop-up, set the **Product install location** as follows:
   - Windows **C:\IBM\AppServer**
   - Linux **/opt/IBM/AppServer**
10. Click **Next**.
11. If the **WebSphere Application server environments** pop-up appears, select **Application Server**, and click **Next**.
12. At the **Enable Administrative Security** pop-up, clear the **Enable administrative security** option, and click **Next**.
13. At the **Installation Summary** pop-up, click **Next** (this will take some time).
14. At the **Installation Results** pop-up, clear the **Launch the First Steps console** option, and click **Finish**.

2.6 Disable the WebSphere Application Server automatic startup for Windows

The core installation of WebSphere Application Server V6.1 creates a base profile that starts automatically when the system is started or restarted. This must be disabled because it causes TCP/IP port conflicts with the test server you will define in section “3.5 Create the ExperienceJ2EE application server profile” on page 51.

The base service name, which you will use to delete the service, has the same value as the profile node name.

1. Open the following file to determine the profile node name:

   ```
   C:\IBM\AppServer\profiles\AppSrv01\logs\AboutThisProfile.txt
   ```

   The node name is listed in the highlighted line:

   ```
   Application server environment to create: Application server
   Location: C:\ibm\appserver\profiles\AppSrv01
   Disk space required: 200 MB
   Profile name: AppSrv01
   Make this profile the default: True
   Node name: expJ2EENode01
   Host name: expJ2EE.demo.com
   Enable administrative security (recommended): False
   ...
   ...
   ```

2. Open a command prompt.
3. Run the following command to remove the startup service (where xxxxxxxx is replaced with the profile node name/base service name as determined in step 1):

   ```
   C:\IBM\AppServer\bin\wasservice -remove xxxxxxxx
   ```

   For example, if the profile node name was **expJ2EENode01** the command and resulting output are as follows:

   ```
   C:\IBM\AppServer\bin\wasservice -remove expJ2EENode01
   Remove Service: expJ2EENode01
   Successfully removed service
   ```

   Alternative: If you have difficulty running the **`wasservice`** command, you could instead disable the service via the Windows Service view. In the example in step #3, where the base node name was expJ2EENode01, the full service name in the Windows Services view would be the following:

   ```
   IBM WebSphere Application Server V6.1 - expJ2EENode01
   ```

2.7 Install fixes for WebSphere Application Server V6.1

WebSphere Application Server V6.1 fixes are packaged in Fix Packs and are installed using the Update Installer utility. The Fix pack and the utility are available at the WAS V6.1 support page, as shown in Table 2-3.

> **Attention:** This book was created and verified using the WebSphere Application Server V6.1.0 Fix Pack 2 (6.1.0.2).
>
> During the final editing, WebSphere Application Server V6.1.0 Fix Pack 3 was released on November 13, 2006.
>
> The information and instructions in this book should still be accurate with Fix Pack 3, but they have not been explicitly verified.

Table 2-3 WAS V6.1 downloadable fixes

Service package	Download location	File size
Update installer	Install Instructions: `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tins_updi_install.html`	n/a
	Windows Update Installer: `ftp://ftp.software.ibm.com/software/websphere/appserv/support/tools/UpdateInstaller/6.1.x/WinIA32/download.updii.6100.windows.ia32.zip`	72.7 mb
	Linux Update Installer: `ftp://ftp.software.ibm.com/software/websphere/appserv/support/tools/UpdateInstaller/6.1.x/LinuxIA32/download.updii.6100.linux.ia32.zip`	72.5 mb

Service package	Download location	File size
Fix Pack 2 (V6.1.0.2)	Readme: http://www.ibm.com/support/docview.wss?rs=180&uid=swg27008308	n/a
	Windows Fixes ftp://ftp.software.ibm.com/software/websphere/appserv/support/fixpacks/was61/cumulative/cf6102/WinX32/6.1.0-WS-WAS-WinX32-FP0000002.pak	252 mb
	Linux Fixes: ftp://ftp.software.ibm.com/software/websphere/appserv/support/fixpacks/was61/cumulative/cf6102/LinuxX32/6.1.0-WS-WAS-LinuxX32-FP0000002.pak	252 mb

2.7.1 Install the Update Installer

1. Download and extract the Update Installer:
   - Windows **download.updii.6100.windows.ia32.zip**
   - Linux **download.updii.6100.linux.ia32.zip**
2. Open a command line—appropriate for your operating system.
3. Change to the directory where you expanded the package, and execute:
   - Windows **.\UpdateInstaller\install**
   - Linux **./UpdateInstaller/install**
4. At the **Installation wizard for the Update Installer** pop-up, click **Next**.
5. At the **Software License Agreement** pop-up, select **I accept both the IBM and non-IBM terms**, and click **Next**.
6. At the **System prerequisites check** pop-up, click **Next**.
7. Linux At the **A Non-root/Non-Administrator user is detected** pop-up, click **Next**.
8. At the **Installation directory** pop-up, set the **Directory Name** as follows, and then click **Next**:
   - Windows **C:\IBM\UpdateInstaller**
   - Linux **/opt/IBM/UpdateInstaller**
9. At the **Installation summary** pop-up, click **Next**.

10. At the **Installation complete** pop-up, perform the following:

    a. Select **Launch IBM Update Installer for WebSphere software on exit**.

    b. Click **Finish**.

2.7.2 Install the fix pack

1. Switch to the open Update Installer. If the Update Installer is NOT open, open a command prompt line, and execute:
   - Windows **C:\IBM\UpdateInstaller\update.bat**
   - Linux **/opt/IBM/UpdateInstaller/update.sh**
2. At the **Welcome...** pop-up, click **Next**.
3. At the **Product selection** pop-up, accept the following default for **Directory name**, and click **Next**:
   - Windows **C:IBM\AppServer**
   - Linux **/opt/IBM/AppServer**
4. At the **Select the maintenance operation** pop-up, select **Install maintenance package**, and click **Next**.
5. At the **Maintenance package selection** pop-up, perform the following:
   - Click **Browse...**, and then at the resulting pop-up, navigate to the directory where you downloaded the Fix Pack.
   - Select the appropriate Fix Pack:
     - Windows **6.1.0-WS-WAS-WinX32-FP0000002.pak**
     - Linux **6.1.0-WS-WAS-LinuxX32-FP0000002.pak**
   - Select **Open**.
   - Click **Next**. Allow a few minutes delay as the update installer verifies the Fix Pack.
6. At the **The following maintenance package will be installed** pop-up, click **Next**.
7. At the **Success** pop-up, click **Finish**.
8. Optional step: Perform the following to verify that the update was successful:
   - Open a command prompt line, and execute the following:
     - Windows **C:\IBM\AppServer\bin\versionInfo**
     - Linux **/opt/IBM/AppServer/bin/versionInfo.sh**
   - Look for the following output (where 6.1.0.2 indicates that the Fix Pack was applied):

```
...

...

Installed Product
--------------------------------------------------------------
Name                      IBM WebSphere Application Server
Version                   6.1.0.2
ID                        BASE
Build Level               cf20633.22
Build Date                8/18/06

---------------------------------------------------------------
End Installation Status Report
---------------------------------------------------------------
```

2.8 Install the base WebSphere Application Server Toolkit V6.1

Perform the following steps to install AST V6.1 with the configuration used in this publication:

1. Start the installation program from the WebSphere Application Server Toolkit image directory (or from the product CD):
   - Windows **.\install.exe**
   - Linux **./install**
2. At the **Welcome …** pop-up, click **Next**.
3. At the **Software License Agreement** pop-up, select **I accept both the IBM and the non-IBM terms**, and click **Next**.
4. At the **Installation destination** pop-up, set the **Directory Name** as follows, and click **Next**:
   - Windows **C:\IBM\AST**
   - Linux **/opt/IBM/AST**
5. At the **Please read the summary information below** pop-up, click **Next**. Allow a delay in time to install the product.
6. After the product installation completes, click **Finish**.

2.9 Install fixes for WebSphere Application Server Toolkit V6.1

You can install fixes through the internet or through downloadable ZIP files. Use the internet to perform the following instructions:

1. Launch the AST V6.1 Product Updater:
   - Windows Programs menu: **Programs → IBM WebSphere → Application Server Toolkit V6.1 → Rational Product Updater**
   - Windows Command line: **C:\IBM\AST\updater\eclipse\rpu.bat**
   - Linux Command line: **/opt/IBM/AST/updater/eclipse/rpu.sh**
2. From the **IBM Rational Product Updater** pop-up, select the **Installed Products** tab.
3. Click **Find Updates**. You will see a **Searching** pop-up that may take a few minutes to complete and then it automatically closes.

HTTP Proxy:

If your location uses an HTTP proxy (not a socks server!) you can set the Product Updater to utilize this proxy.

From the **IBM Rational Product Updater** action bar, select **Preferences → Proxy Settings**:

- Select **Enable HTTP Proxy connection.**
- Enter the **HTTP Proxy host address** (host name only -- no http://).
- Enter the **HTTP Proxy host port.**
- Click **OK**.
- Back at the main pop-up, click **Find Updates.**

4. From the **IBM Rational Product Updater** pop-up (which should have automatically switched to the Updates tab), click **Install Updates**.

Latest fix pack:

The latest available fixpack when this redbook was published was the IBM WebSphere Application Server Toolkit Fix Pack 1.

5. From the **License Agreement** pop-up, select **I accept the terms in the license agreement**.

6. Click **OK**. This starts the download and installation of the updates, which may take some time.
7. After the **Progress Information** pop-up closes, exit the **IBM Rational Product Updater**.

If you cannot access the fixes directly over the internet through these steps, download the latest fixes and install instructions from the Web sites shown in Table 2-4.

Table 2-4 AST V6.1 downloadable fixes

Service package	Download location	File size
Fix Pack 1 (V6.1.0.1)	Install Instructions: ftp://ftp.software.ibm.com/software/websphere/appserv/support/tools/AST/local/fixpacks/ast61/fp6101/html/install.html	9 kb
	Fixes: ftp://ftp.software.ibm.com/software/websphere/appserv/support/tools/AST/local/fixpacks/ast61/fp6101/ast61fixpack1.zip	73.25 mb

If these download locations are no longer valid please check the WAS V6.1 support page for the latest updates for AST V6.1:

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

3

Configure the development environment

In this chapter you configure the AST V6.1 development environment to support the Experience J2EE! scenario.

3.1 Learn!

Figure 3-1 shows an overview of the Application Server Toolkit.

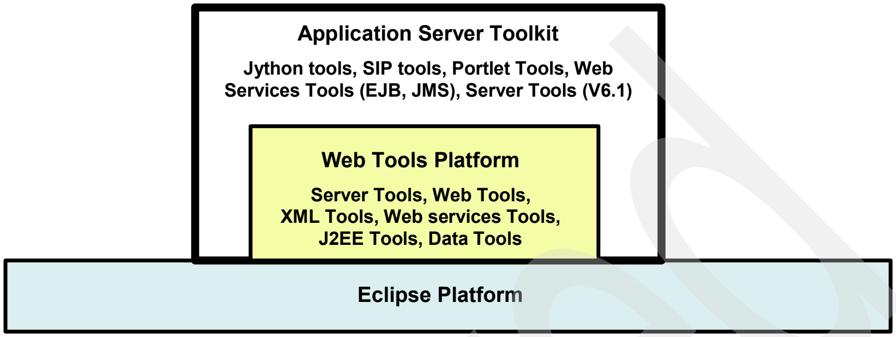

Application Server Toolkit
Jython tools, SIP tools, Portlet Tools, Web Services Tools (EJB, JMS), Server Tools (V6.1)
Web Tools Platform
Server Tools, Web Tools, XML Tools, Web services Tools, J2EE Tools, Data Tools
Eclipse Platform

Figure 3-1 WebSphere Application Server 6.1 tooling overview

AST V6.1 is a component of the WebSphere Application Server V6.1 software package that provides a basic integrated development environment (IDE) for WAS V6.1.

AST V6.1 is based on evolving open-standard IDE initiatives:

- Eclipse (v3.12) provides the overall IDE framework, Java (Java 5 or lower) development, and test tools.

  The following quote is from the AST V6.1 online help: "*Eclipse is an award-winning, open source platform for the construction of powerful software development tools and rich desktop applications. Leveraging the Eclipse plug-in framework to integrate technology on the desktop saves technology providers time and money by enabling them to focus their efforts on delivering differentiation and value for their offerings. Eclipse is a multi-language, multi-platform, multi-vendor supported environment that it is built by an open source community of developers and it is provided royalty-free by the Eclipse Foundation. Eclipse is written in the Java language, includes extensive plug-in construction toolkits and examples, and can be extended and deployed on a range of desktop operating systems including Windows, Linux, QNX and Macintosh OS X.*"

- Web Tools Platform (WTP) (v1.0.2) provides Eclipse-based tooling for developing J2EE applications.

  The following quote is from the Web Tools Platform Web site: "*The Eclipse Web Tools Platform (WTP) project extends the Eclipse platform with tools for developing J2EE Web applications. The WTP project includes the following tools: Source editors for HTML, Javascript, CSS, JSP, SQL, XML, DTD, XSD,* *and WSDL; graphical editors for XSD and WSDL; J2EE project natures, builders, and models and a J2EE navigator; a Web service wizard and explorer, and WS-I Test Tools; and database access and query tools and models*."

- AST improves the binding between WTP and WAS V6.1 by providing for the execution of WAS V6.1 administrative scripts, enhanced Web services tools, graphical editors for deployment descriptor files, Session Initiation Protocol (SIP) support, and unit test and debugging tools.

The combined Eclipse/WTP/AST V6.1 package provides both development and test capabilities:

- Users can create and edit J2EE specific artifacts.
- Users can test the artifacts against J2EE test servers (in this case WAS V6.1).

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

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.

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`

- AST V6.1 on-line information center (also installed locally with AST V6.1)

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.welcome.ast.doc/topics/astoverview.html`

- IBM Education Assistant: Application Server Toolkit

  `http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.was_v6/was/6.1/DevelopmentTools/WASv61_ASTOverview/player.html`

3.2 Extract the ExperienceJ2EE.zip samples file

The ExperienceJ2EE samples ZIP contains various files that are used in the scenarios (Table 3-1).

Table 3-1 ExperienceJ2EE.zip contents

File	Description
DonateInterchange.zip	AST V6.1 project interchange file containing the completed J2EE core application for those wishing to skip the core J2EE application scenarios and focus on the Web services and messaging (JMS) scenarios. This file is used in Chapter 12, “Alternative Import the core J2EE application” on page 299.
employee.ddl	Vacation table definitions in data definition language format (DDL). Used in “4.2 Create the Vacation application database” on page 72.
experiencej2ee_snippets.xml	Code snippets used to create various portions of the J2EE application. Used in “3.8 Import the sample code snippets” on page 64.

1. Ensure that you are logged on to the system as the user that will be used for the remainder of this book:
   - Windows User with local administrator privileges.
   - Linux The root or non-root user that installed the products in the previous chapter.
2. Extract the samples file **ExperienceJ2EE.zip**:
   - Windows **unzip** to directory **C:\ExperienceJ2EE**
   - Linux **unzip** to directory **$HOME/ExperienceJ2EE**

   where $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 AST V6.1 workspace

AST V6.1 (and any Eclipse based product) operates against a collection of resources called a workspace that provides a contained and isolated environment. A workspace typically contains the following:

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

A resource can represent a discreet 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 plugins. Products like AST V6.1 come with a built-in set of plugins, but you can also download and install plugins from other sources as well. The Eclipse Foundation manages the following plug-in download site:

  `http://www.eclipseplugincentral.com`

- 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 run time test environments such as J2EE 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.

In this section you create a single workspace to use for all scenarios in this book.

1. Start AST V6.1:
   - Windows Programs menu: **Programs** → **IBM WebSphere** → **Application Server Toolkit V6.1** → **Application Server Toolkit**
   - Windows Command line: **C:\IBM\AST\ast.exe**
   - Linux Command line: **/opt/IBM/AST/ast&**
2. At the **Workspace Launcher** pop-up, do **NOT** select **Use this as the default and do not ask again**. Set the **Workspace** path as follows, and click **OK**.
   - Windows **C:\Workspace\ExperienceJ2EE**
   - Linux **$HOME/workspace/ExperienceJ2EE**

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

Behind the scenes:

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.

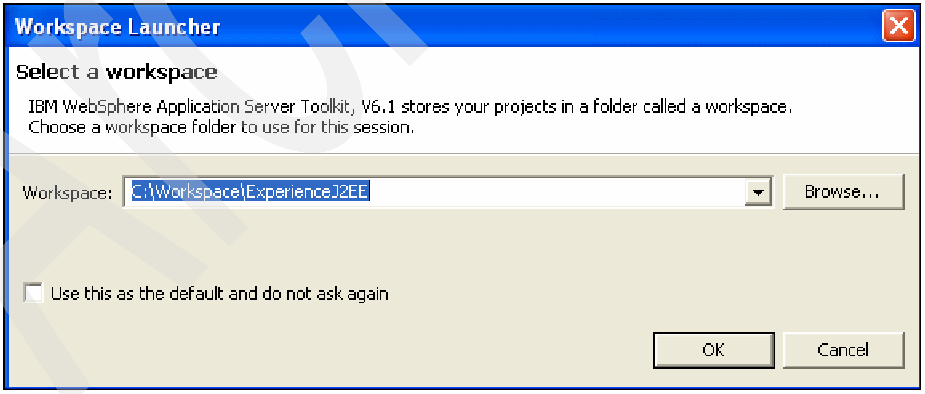
If you accidentally set this, you can re-enable the workspace prompt:

- From within an open AST V6.1workspace, go to the application action bar, select **Window** → **Preferences**
- At the **Preferences** pop-up perform the following:
  - On the left, select **General** → **Startup and Shutdown**.
  - On the right, select **Prompt for workspace on startup**, and click **OK**.

The workspace selection prompt should appear the next time you start AST V6.1.

Accepting the default workspace on Windows under *Document and Settings* causes problems when running some wizards (like creating a Web service client) due to spaces in the directory path.

If the indicated folder does not exist, AST V6.1 creates it automatically and initializes the workspace by adding a *.metadata* folder where meta-information will be stored.



Workspace Launcher

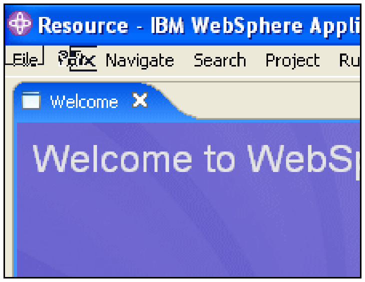
Select a workspace

IBM WebSphere Application Server Toolkit, V6.1 stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

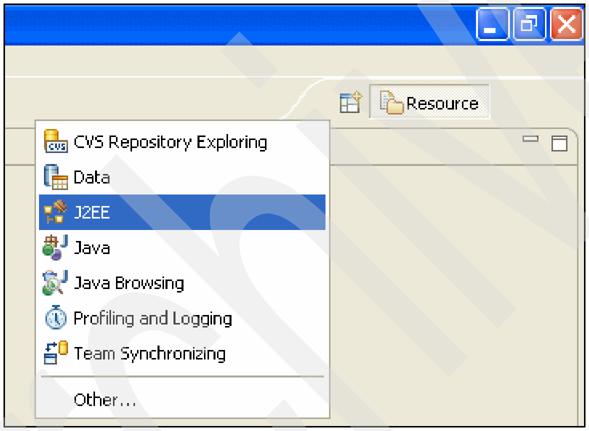
Workspace: C:\Workspace\ExperienceJ2EE

□ Use this as the default and do not ask again

3. After the workspace initializes, close the **Welcome** pane by clicking the X next to Welcome. Note that you can re-open it easily if required at a later time by selecting **Help** → **Welcome** from the menu bar.



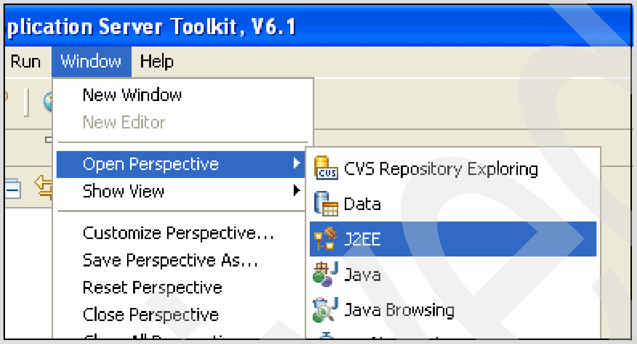
4. Switch to the J2EE Perspective. Click the **Select Perspective** icon [ ] located at the upper right, and select **J2EE** from the pull-down.



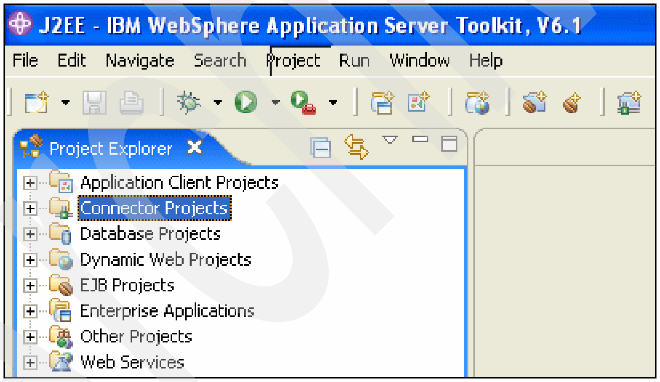
Off course?

If J2EE is not visible, select **Other** at the bottom of the pop-up, and then select **J2EE** from the resulting **Select Perspective** pop-up.

Alternative: You can also switch perspectives from the action bar by selecting **Window** → **Open Perspective** → **J2EE**.



5. The title bar in the upper left should now display **J2EE - IBM WebSphere Application Server Toolkit, V6.1**.



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 J2EE development), only showing views and capabilities that apply for that role.

3.4 Configure the WAS V6.1 server runtime definition

Unlike previous IBM development tools (such as Rational Application Developer V6.x), AST V6.1 does not ship with an actual test environment. Instead, it contains a 'stub' for a WAS V6.1 test environment that can be replaced with a pointer to an actual WAS V6.1 installation.

Why this change? There are several advantages that led to this:

- The AST V6.1 software package is smaller. AST V6.1 (since it just contains the tooling and not the runtime) fits on a single CD. This may not seem significant with AST V6.1, which supports a single runtime. But consider Rational Application Develop V6.x that ships multiple tests environments and fits on nine CDs.
- Developers that wish to have both the test environments and the external environments on the same system now have a single install copy instead of two.
- When developers apply fixes for WAS V6.1 and AST V6.1, there is no longer any confusion regarding which fixes to apply where: Apply the WAS V6.1 fixes to the WAS V6.1 installation directories, and apply the AST V6.1 fixes to the AST V6.1 directories.

AST V6.1 is actually part of the overall WAS V6.1. You purchase one package (WAS V6.1) that contains both the tooling (AST V6.1) and the test environment (WAS V6.1 server component).

IBM sells a development version of WAS V6.1 called WebSphere Application Server for Developers V6.1. This provides a limited license (and thus lower cost) version of WAS V6.1 that can be used to develop and test applications.

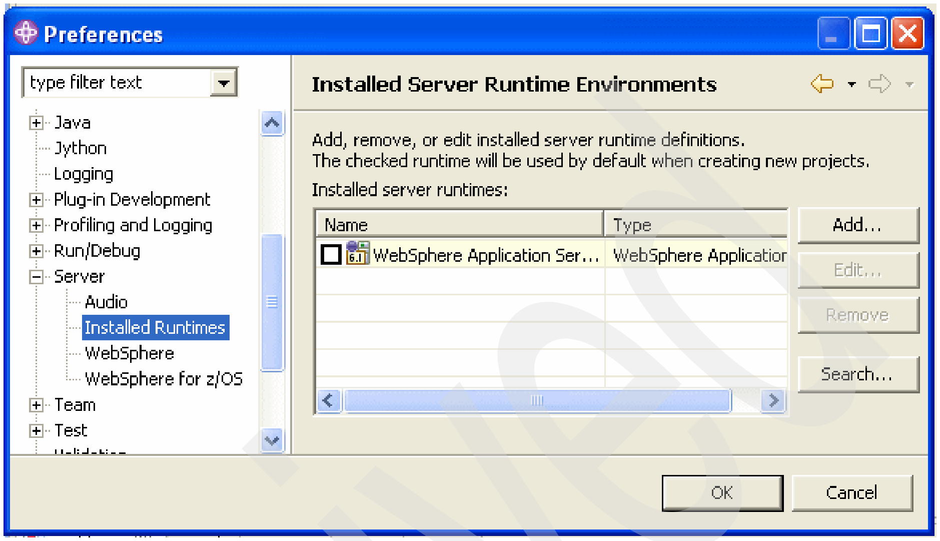
The primary disadvantage is that the developer must run two separate installs for AST V6.1 and for WAS V6.1 instead of just one install for Rational Application Developer.

The user community is the ultimate arbiter in deciding whether this is a positive change, or not. The only certainty (in the author's opinion) is that this will be the source of spirited debate.

To configure a test server perform the following steps:

1. Navigate to the **Installed Runtimes** preferences:
   – From the application action bar, select **Window** → **Preferences**.

– At the **Preferences** pop-up, scroll down and select **Server** → **Installed Runtimes**. The **Installed Server runtime Environments pane** will display on the right.



Installed Server Runtime Environments

Add, remove, or edit installed server runtime definitions.
The checked runtime will be used by default when creating new projects.

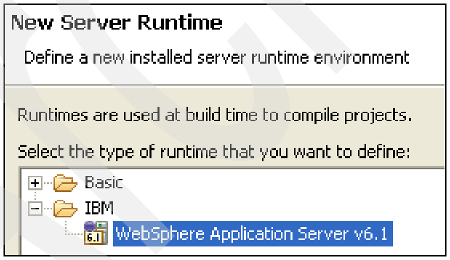
Installed server runtimes:

Name	Type
□ WebSphere Application Ser...	WebSphere Applicatior

2. Add the WAS V6.1 definition by performing the following actions:

   – On the right of the **Installed Server Runtime Environments** pane, left-click **Add...**

   – At the **New Server Runtime** pop-up, expand to select **IBM** → **WebSphere Application Server V6.1**.



New Server Runtime

Define a new installed server runtime environment

Runtimes are used at build time to compile projects.

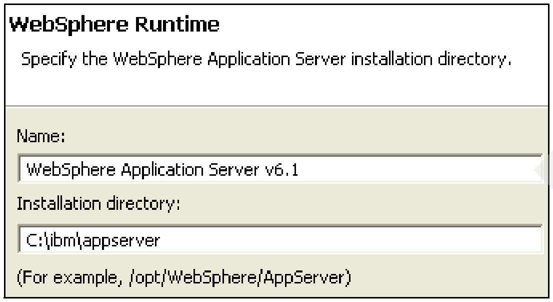
Select the type of runtime that you want to define:

   – Click **Next**.

   – At the **WebSphere Runtime** pop-up, set the **Installation directory**:

     - Windows **C:\IBM\AppServer**
     - Linux **/opt/IBM/AppServer**

   – Click **Finish**.



WebSphere Runtime

Specify the WebSphere Application Server installation directory.

Name:

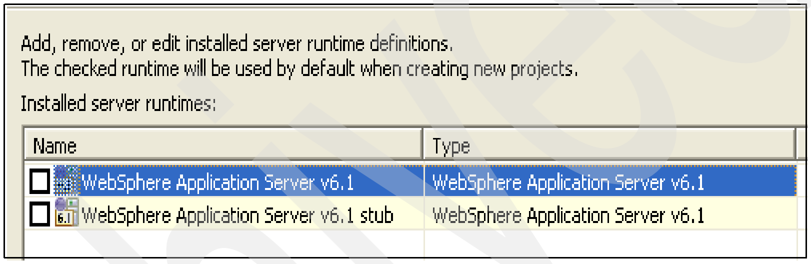
WebSphere Application Server v6.1

Installation directory:

C:\ibm\appserver

(For example, /opt/WebSphere/AppServer)

3. At the **Preferences** pop-up, note the new entry (for WebSphere Application Server v6.1, in contrast to the existing WebSphere Application Server v6.1 **stub**). You may need to expand the **Name** column to see the complete string.



Add, remove, or edit installed server runtime definitions.
The checked runtime will be used by default when creating new projects.

Installed server runtimes:

Name	Type
□ WebSphere Application Server v6.1	WebSphere Application Server v6.1
□ WebSphere Application Server v6.1 stub	WebSphere Application Server v6.1

4. Leave the **Preferences** pop-up open. You will use it in the next section.

3.5 Create the ExperienceJ2EE application server profile

The runtime environment configuration for a WebSphere Application Server V6.1 (WAS V6.1) 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. WAS V6.1 ships with a default profile (normally called AppSrv01).

Each profile also contains a copy of the standard WAS V6.1 commands (such as **`startServer`**, **`stopServer`**, and **`wsadmin`**) that by default operate against that profile. The commands in the default command directory (C:\IBM\AppServer\bin) operate against the default profile, unless overridden by the profileName parameter.

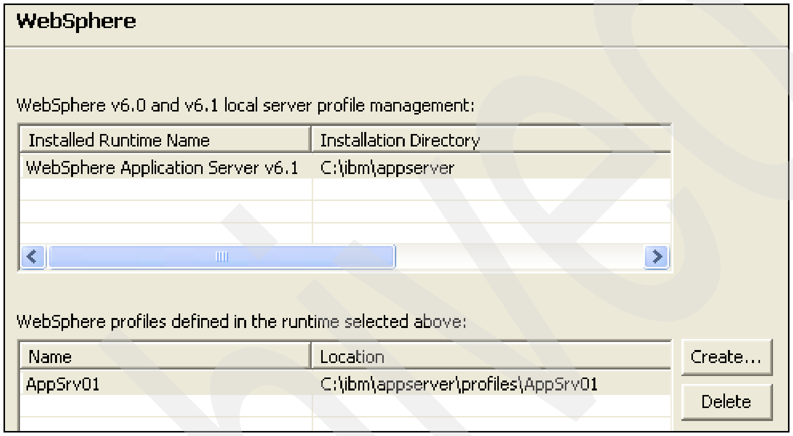
When working with the Application Server Toolkit V6.1 (AST V6.1) and other Eclipse-based development environments, it is advisable to have a separate profile for each significant application and development environment.

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, perform the following steps:

1. From the open **Preferences** pop-up, select **Server** → **WebSphere**.
2. On the right in the **WebSphere** pane, next to the **WebSphere profiles defined... list**, left-click **Create...** to start the wizard.



WebSphere

WebSphere v6.0 and v6.1 local server profile management:

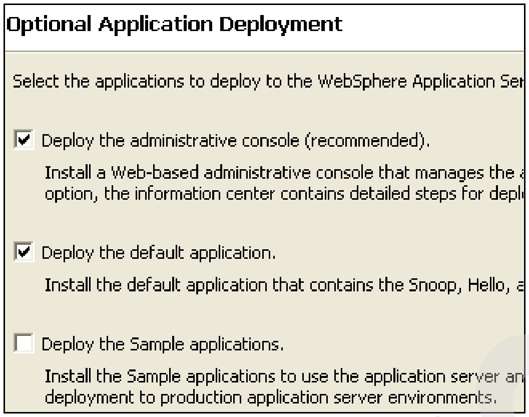
Installed Runtime Name	Installation Directory
WebSphere Application Server v6.1	C:\ibm\appserver

WebSphere profiles defined in the runtime selected above:

Name	Location
AppSrv01	C:\ibm\appserver\profiles\AppSrv01

3. At the **Welcome to the Profile Management tool** pop-up, click **Next**.
4. At the **Environment Selection** pop-up, ensure that **Application Server** is selected, and click **Next**.
5. At the **Profile Creation Options** pop-up, select **Advanced profile creation**, and click **Next**.

6. At the **Optional Application Deployment pop-up,** accept the default selections, as shown in the following image. Click **Next**.



Optional Application Deployment

Select the applications to deploy to the WebSphere Application Ser

☑ Deploy the administrative console (recommended).

Install a Web-based administrative console that manages the a option, the information center contains detailed steps for depl

☑ Deploy the default application.

Install the default application that contains the Snoop, Hello, a

□ Deploy the Sample applications.

Install the Sample applications to use the application server an deployment to production application server environments.

7. At the **Profile Name and Location** pop-up, perform the following actions:

   – **Profile Name**: **ExperienceJ2EE**

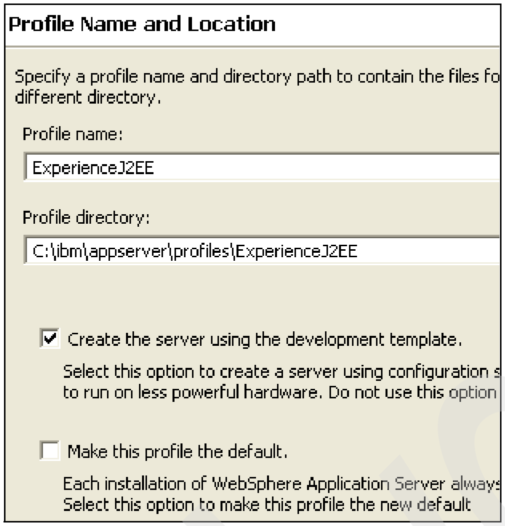
   – Profile Directory:

     - Windows **C:\IBM\AppServer\profiles\ExperienceJ2EE**
     - Linux **/opt/IBM/AppServer/profiles/ExperienceJ2EE**

   – Select **Create the server using the development template**.

   – Do **NOT** select **Make this profile the default**.

   – Click **Next**.



Profile Name and Location

Specify a profile name and directory path to contain the files fo
different directory.

Profile name:
ExperienceJ2EE

Profile directory:
C:\ibm\appserver\profiles\ExperienceJ2EE

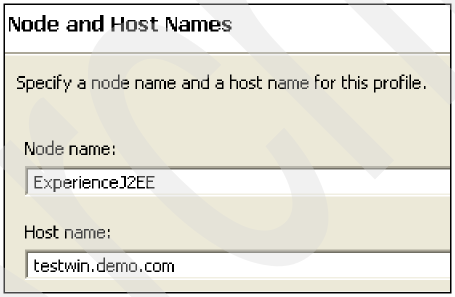
☑ Create the server using the development template.

Select this option to create a server using configuration s
to run on less powerful hardware. Do not use this option

□ Make this profile the default.

Each installation of WebSphere Application Server always
Select this option to make this profile the new default

8. At the **Node and host names** pop-up, do the following:
   - Change the **Node Name** to **ExperienceJ2EE**.
   - Accept the default value for the **Host name**.
   - Click **Next**.



Node and Host Names

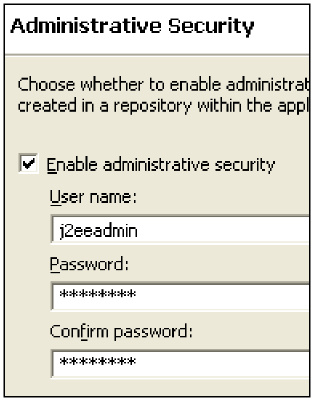
Specify a node name and a host name for this profile.

Node name:
ExperienceJ2EE

Host name:
testwin.demo.com

9. At the **Administrative Security** pop-up, perform the following tasks:
   - Select **Enable administrative security**.
   - **User name**: **j2eeadmin**
   - Set **Password** and **Confirm password** to any password value.

- Click **Next**.



Administrative Security

Choose whether to enable administrat
created in a repository within the appl

☑ Enable administrative security

User name: j2eeadmin

Password: ********

Confirm password: ********

Behind the scenes:

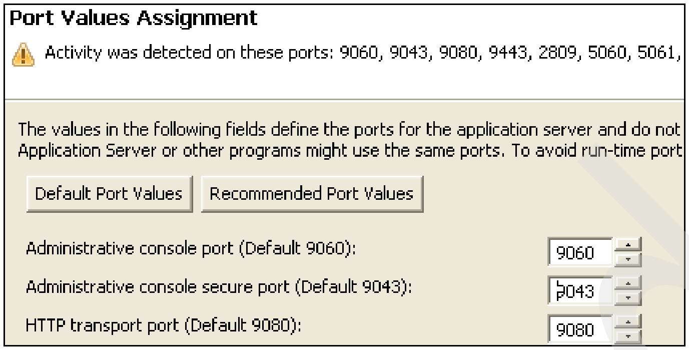
This option configures security for using the WAS V6.1 administrative interfaces: The Web based WebSphere Administrative Console, the **`wsadmin`** command line interface, and the Java Management Extensions (JMX™) interface. This ensures that only authorized users are able to access the administrative 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.
- WAS V6.1 allows a progressive level of enabling security:
  - Administrative security only (as configured at this point).
  - Administrative Security and J2EE application security, which are used to control access to J2EE 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.

  We will implement J2EE application security in Chapter 11, “Implement core security” on page 257.

10. At the **Port value assignment** pop-up, click **Default Port Values** to change all the values on the right back to the defaults.

11. Click **Next**.



Port Values Assignment

⚠ Activity was detected on these ports: 9060, 9043, 9080, 9443, 2809, 5060, 5061,

The values in the following fields define the ports for the application server and do not
Application Server or other programs might use the same ports. To avoid run-time port

Administrative console port (Default 9060): 9060

Administrative console secure port (Default 9043): 9043

HTTP transport port (Default 9080): 9080

Behind the scenes:

Note the warning message **Activity was defected on these ports: 9060, 9043...** In this case, this indicates that other WAS V6.1 profiles exist that use these ports.

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

This is also the reason why you disabled the automatic startup of the default server in section "2.6 Disable the WebSphere Application Server automatic startup for Windows" on page 33.

12. Windows At the **Windows Service Definition** pop-up, clear **Run the application server process as a Windows service**, and click **Next**.
13. At the **Web Server Definition** pop-up, leave **Create a Web server definition** DISABLED, and click **Next**.
14. At the **Profile Creation Summary** pop-up, click **Create**. This will take a few minutes.
15. At the **Profile Creation Complete** pop-up, clear **Launch the First steps console**.
16. Click **Finish**.

Behind the scenes:

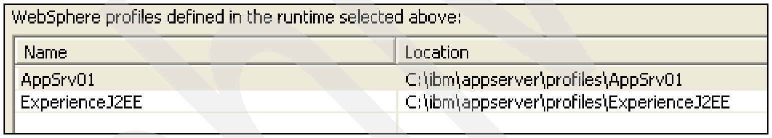
The **First Steps console** provides several tools for WAS V6.1 beginners:

- **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 WAS V6.1 runtime profile definitions.
- **Information center** available on the Internet:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp`
- **Migration wizard**, for migrating profiles from earlier versions of WebSphere Application Server.

17.At the **Preferences** pop-up, note that the new profile is listed.



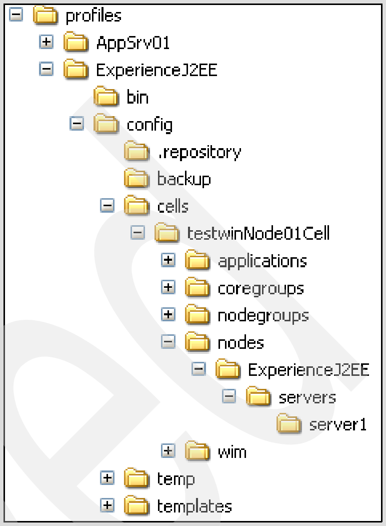
WebSphere profiles defined in the runtime selected above:

Name	Location
AppSrv01	C:\ibm\appserver\profiles\AppSrv01
ExperienceJ2EE	C:\ibm\appserver\profiles\ExperienceJ2EE

18.Click **OK** to close the **Preferences** pop-up.

Behind the scenes:

The complete profile definition is created in the file system in the <WAS V6.1>\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 ExperienceJ2EE) 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 using the graphical profile creation tool used earlier or through the **manageprofiles** (.sh on Linux) command found in the <WASDIR>\bin directory.

```
manageprofiles -delete -profileName ExperienceJ2EE
```

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.

3.6 Create the ExperienceJ2EE Server @ localhost test server definition

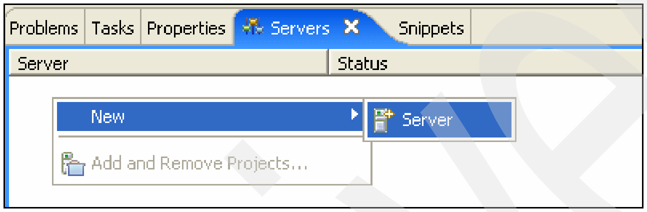
The workspace can contain the definitions of multiple test servers that identify the runtime environments to which you wish to deploy, test, and debug applications.

AST V6.1 supports a single type of test server, WebSphere Application Server V6.1. Other IBM Eclipsed-based products, such as Rational Application Developer and WebSphere Integration Developer, can support multiple types of test servers: WebSphere Application Server, WebSphere Process Server, WebSphere Enterprise Service Bus, WebSphere Portal Server, and Apache Tomcat.

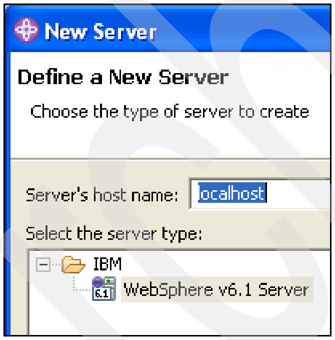
Test servers can be local, meaning that their executables are installed with AST V6.1 and run on the same local system. Alternately, test servers can be remote, meaning that their executables are installed and run on a different system or a hostname other than localhost or 127.0.0.1.

In this section you define a local WAS V6.1 test server that references the ExperienceJ2EE profile created earlier.

1. Switch to the **Servers** view in the lower-right pane by clicking the tab named **Servers**. Right-click the white space, and select **New** → **Server**.



2. At the **Define a new Server** pop-up, accept the defaults, and click **Next**.


New Server

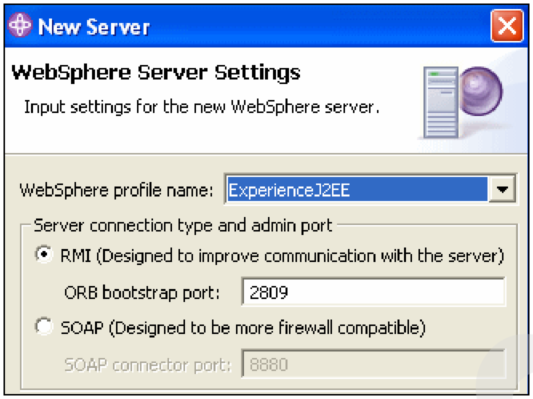
Define a New Server

Choose the type of server to create

Server's host name: localhost

Select the server type:

3. At the **WebSphere Server Settings** pop-up, select **ExperienceJ2EE** from the **WebSphere profile name** pull-down, and click **Finish**.



New Server

WebSphere Server Settings

Input settings for the new WebSphere server.

WebSphere profile name: ExperienceJ2EE

Server connection type and admin port

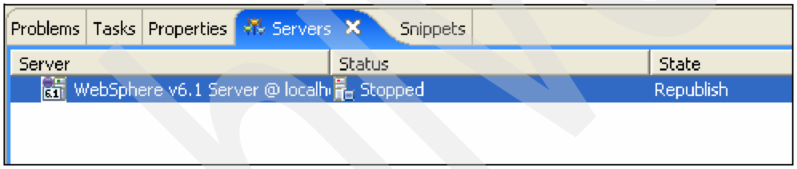
● RMI (Designed to improve communication with the server)

ORB bootstrap port: 2809

○ SOAP (Designed to be more firewall compatible)

SOAP connector port: 8880

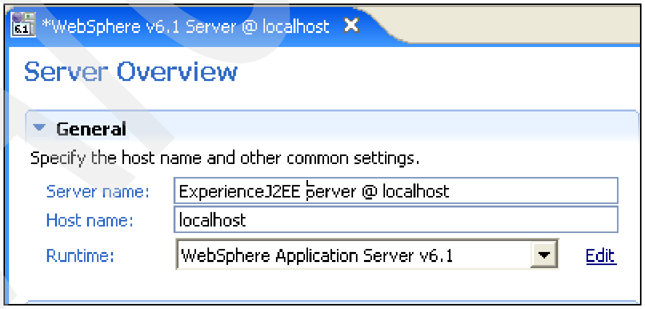
4. The **Servers** view should now contain the new entry. Double-click on this entry -- **WebSphere v6.1 Server @ localhost**. You may have to scroll over to see the complete name.



Server	Status	State
WebSphere v6.1 Server @ localh	Stopped	Republish

5. Locate the resulting **WebSphere v6.1 Server @ localhost** editor, in the upper center pane, and perform the following tasks.

   – Access the **General** section, and change the **Server name** field to **ExperienceJ2EE Server @ localhost**.



*WebSphere v6.1 Server @ localhost

Server Overview

General

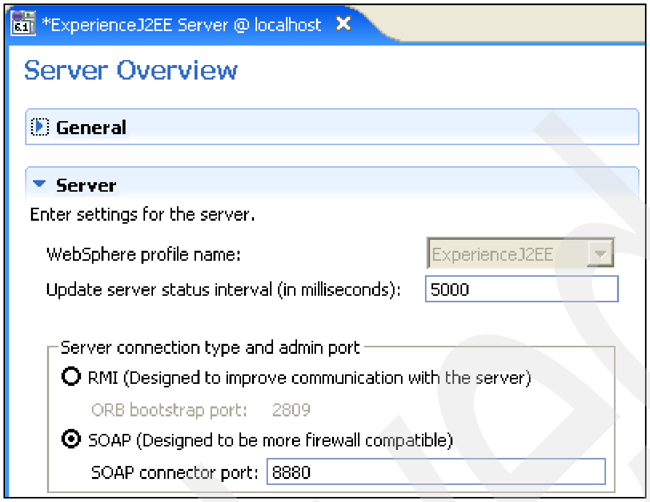
Specify the host name and other common settings.

Server name: ExperienceJ2EE Server @ localhost

Host name: localhost

Runtime: WebSphere Application Server v6.1

– Under the **Server** section, select **SOAP (Designed to be more firewall compatible)**.



*ExperienceJ2EE Server @ localhost

Server Overview

General

Server

Enter settings for the server.

WebSphere profile name: ExperienceJ2EE

Update server status interval (in milliseconds): 5000

Server connection type and admin port

○ RMI (Designed to improve communication with the server)

ORB bootstrap port: 2809

● SOAP (Designed to be more firewall compatible)

SOAP connector port: 8880

Behind the scenes:

WAS V6.1 supports two types of communication with the administrative service: RMI and SOAP. Either works on Windows, but when running with a non-root user on Linux you appear to require the SOAP communication.

Otherwise, when starting the test server, the server status remains as Starting.... even though log files indicate that the server was successfully started. Then after several minutes you receive a pop-up message that states the following:

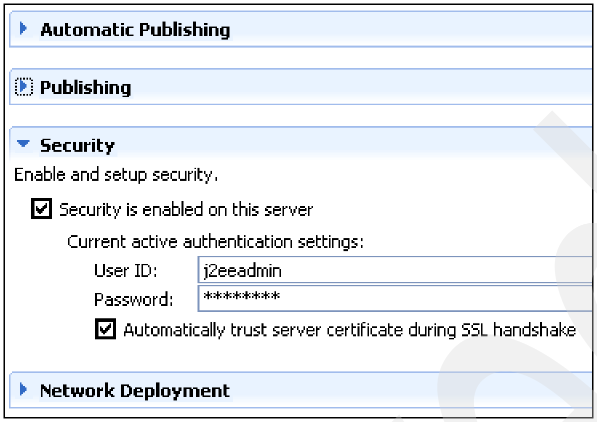
```
Timeout waiting for ExperienceJ2EE Server @ localhost to
start. Server did not start after 210000s
```

Note that the message does indicate 210000 seconds, but when this occurred, the time-out was actually around 210 seconds after the start was issued.

– Under the **Security** section (on the right), perform the following:

- Set **User id** to **j2eeadmin**.

- Set **Password** to the value you set in section “3.5 Create the ExperienceJ2EE application server profile” on page 51.



Enable and setup security.

☑ Security is enabled on this server

Current active authentication settings:

User ID: j2eeadmin

Password: *********

☑ Automatically trust server certificate during SSL handshake

– Save and close the changes using one of the following methods:

- Click ☒ next to the name, and click **Yes** at the **Save Resource** pop-up.
- Ensure that the editor window is active, and select **File** → **Save** and then **File** → **Close**.
- Save the file through a CTL-S key sequence, and close the file through a CTL-F4 key sequence.

The server list is now updated to show **ExperienceJ2EEServer @ localhost**.

3.7 Start the ExperienceJ2EE test server

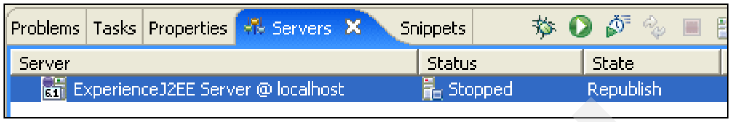
Earlier IBM J2EE development tools required that you restart the test server to pick up changes such as adding JDBC providers and data sources.

With the AST V6.1 and Rational Application Developer V6.0 development tools (and the WebSphere Application Server V6.x runtime), 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.

Some exceptions, which do require restarting the test server, include enabling or disabling security and adding a service bus messaging engine.

1. Switch to the **Servers** view in the lower right pane.

2. Select **ExperienceJ2EE Server @ localhost**.
3. Click the start icon [▶] in the action bar.



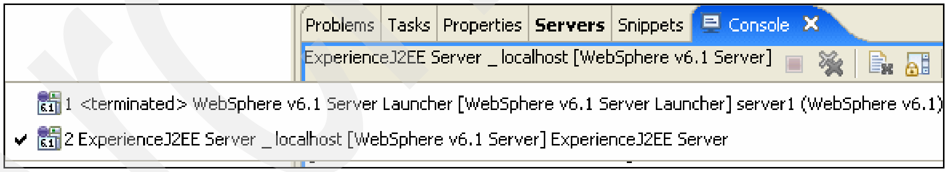
Server	Status	State
ExperienceJ2EE Server @ localhost	Stopped	Republish

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

4. If you receive any firewall prompts, allow access by java or javaw.
5. AST V6.1 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 **ExperienceJ2EE Server @ localhost …** log.
6. Verify that startup occurred normally (for example that there are no red messages), and look for the following line (up 20 to 25 lines from the bottom):

```
[6/19/06 13:41:43:953 CDT] 0000000a WsServerImpl  A   WSVR0001I:
Server server1 open for e-business
```

7. If you do not see the messages described in step 6, use the Console switch selection [🖥 ▾] to ensure that the **Console** view is showing the **ExperienceJ2EE Server @ localhost …** log (which contains the operational messages). The other available selection is the **WebSphere V6.1 Server Launcher…**, which shows the messages generated by the batch file that starts the server.



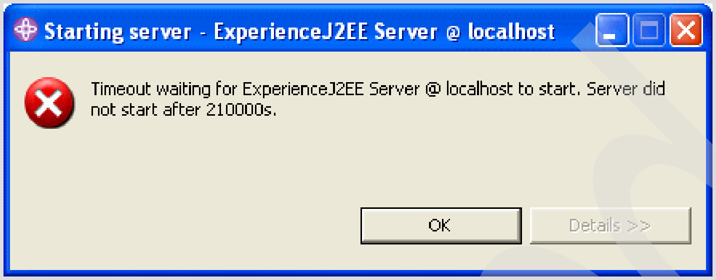
ExperienceJ2EE Server _ localhost [WebSphere v6.1 Server]

1 <terminated> WebSphere v6.1 Server Launcher [WebSphere v6.1 Server Launcher] server1 (WebSphere v6.1)

✔ 2 ExperienceJ2EE Server _ localhost [WebSphere v6.1 Server] ExperienceJ2EE Server

Off course?

You may see the following pop-up message if your system performance is slow or if you do not respond quickly to any firewall prompts:



Starting server - ExperienceJ2EE Server @ localhost

Timeout waiting for ExperienceJ2EE Server @ localhost to start. Server did not start after 210000s.

You can ignore this message if your server started properly as defined in the previous step—meaning that the server status changes to **Started** and that the **Console** view shows the **server1 open for e-business** message.

If this pop-up message appears regularly when starting the server, you can configure AST V6.1 to use a longer server timeout delay.

- From within an open AST V6.1 workspace, go to the application action bar, select **Window** → **Preferences**.
- Perform the following at the **Preferences** pop-up:
  - On the left, select **Server.**
  - On the right, change the **Server timeout delay** setting from **Normal** to **Long** or **Longer**, and click **OK**.

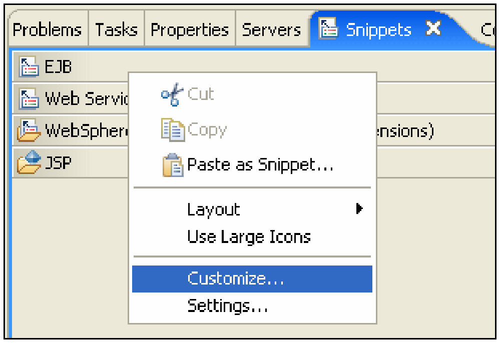
3.8 Import the sample code snippets

AST V6.1 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.

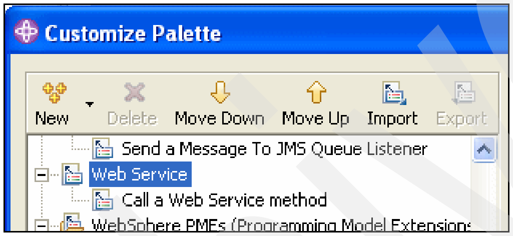
In this case, you use snippets as a simple cut and paste mechanism when adding code to various Java classes and Web pages, instead of manually typing in the code or copying from a text file.

1. Switch to the **Snippets** view in the lower right pane.

2. Right-click anywhere inside the view, and select **Customize…**.



3. At the **Customize Palette** pop-up, click **Import**.


Customize Palette
New
Delete
Move Down
Move Up
Import
Export
Send a Message To JMS Queue Listener
Web Service
Call a Web Service method
WebSphere PMEs (Programming Model Extensions

4. At the directory browser pop-up perform the following:

   – Windows Navigate to the directory **C:\ExperienceJ2EE**, select **experiencej2ee_snippets(.xml)**, and click **Open**.

   – Linux Navigate to the directory **$HOME/ExperienceJ2EE**, select **experiencej2ee_snippets(.xml)**, and click **OK**.

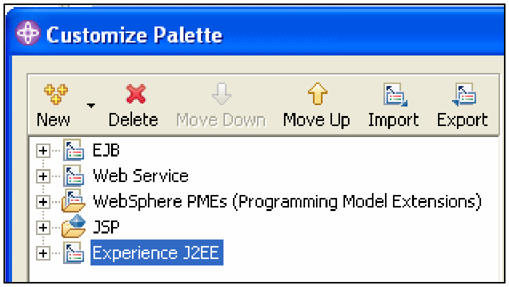
   where $HOME is the home directory for the Linux user. For example, if the Linux user is testuser, the home directory is usually /home/testuser.

5. At the **Customize Palette** pop-up, and move the newly created category to the top of the list, by performing the following:

   – Scroll to the bottom of the list (if not already there).

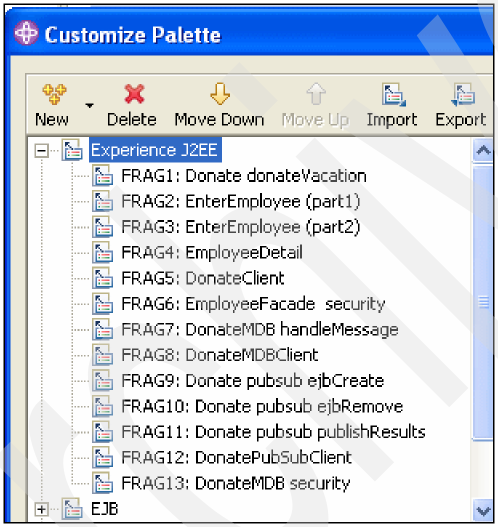
   – Select the **Experience J2EE** category.

– Click **Move Up** until the **Experience J2EE** category is at the top of the list.



EJB
Web Service
WebSphere PMEs (Programming Model Extensions)
JSP
Experience J2EE

6. Expand the **Experience J2EE** category by clicking the plus sign ⊞ located next to it. Now you can view the list of imported snippets for this category, as shown in the following image.



Experience J2EE
FRAG1: Donate donateVacation
FRAG2: EnterEmployee (part1)
FRAG3: EnterEmployee (part2)
FRAG4: EmployeeDetail
FRAG5: DonateClient
FRAG6: EmployeeFacade security
FRAG7: DonateMDB handleMessage
FRAG8: DonateMDBClient
FRAG9: Donate pubsub ejbCreate
FRAG10: Donate pubsub ejbRemove
FRAG11: Donate pubsub publishResults
FRAG12: DonatePubSubClient
FRAG13: DonateMDB security
EJB

7. Click **OK** to close the pop-up.

3.9 Explore!

Figure 3-2 shows the various IDEs and the functions they contain.

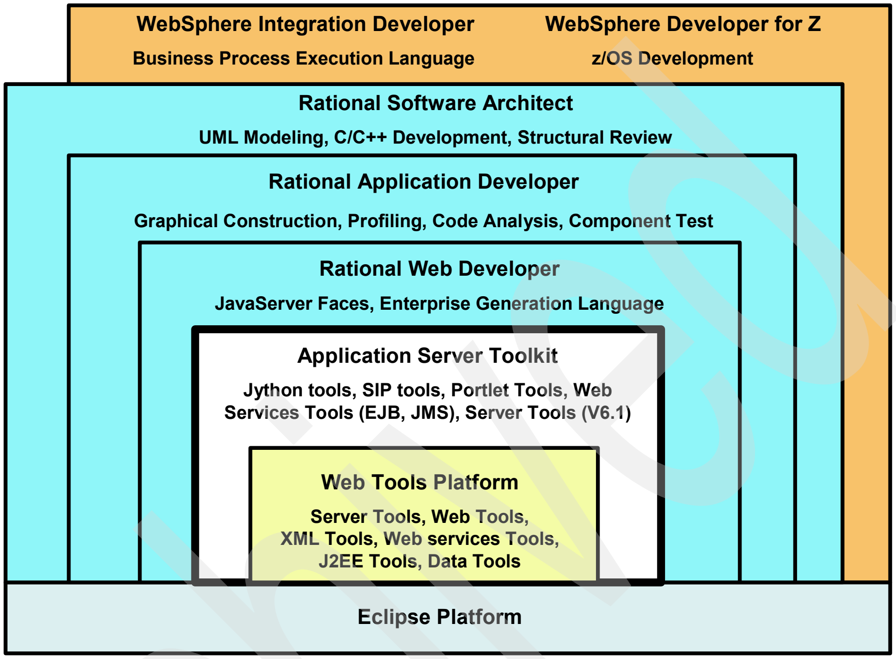

WebSphere Integration Developer
Business Process Execution Language
WebSphere Developer for Z
z/OS Development
Rational Software Architect
UML Modeling, C/C++ Development, Structural Review
Rational Application Developer
Graphical Construction, Profiling, Code Analysis, Component Test
Rational Web Developer
JavaServer Faces, Enterprise Generation Language
Application Server Toolkit
Jython tools, SIP tools, Portlet Tools, Web Services Tools (EJB, JMS), Server Tools (V6.1)
Web Tools Platform
Server Tools, Web Tools, XML Tools, Web services Tools, J2EE Tools, Data Tools
Eclipse Platform

Figure 3-2 IBM integrated development environments

AST V6.1 provides a basic J2EE IDE, but IBM has other products that provide enhanced IDE support:

- Rational Web Developer V6.0 provides an enhanced IDE for Web and Java developers, with additional capabilities such as the following:
  - Visual editors for Web development (HTML, JSP, Struts, JSF).
  - Enterprise Generation Language (EGL).
  - Support for other J2EE servers (including older WebSphere Application Server releases, Apache Tomcat, and BEA WebLogic).
- Rational Application Developer V6.0 provides a comprehensive Java/J2EE oriented IDE, with additional capabilities such as the following:
  - Visual editors for EJB development.
  - Integrated WebSphere Portal development/testing.
  - Basic Unified Modeling Language (UML) support.
  - Code analysis and validation tools (profiling).

  - Support for Service Oriented Architecture (SOA) extensions such Service Data Objects (SDOs).

- Rational Software Architect V6.0 provides 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 J2EE artifacts that then can be completed and extended by AST V6.1, Rational Web Developer and Rational Application Developer.
- WebSphere Integration Developer V6.0 provides for the development and testing of non-J2EE artifacts such as Enterprise Service Bus (ESB) mediations and WS-BPEL based workflows. Reference sections “18.6 Enterprise Service Buses” on page 423 and “18.7 Business Process Execution Language” on page 425 for more information.

AST V6.1 was the only IBM IDE that supported WAS V6.1 when that product was announced on April 11, 2006. However, IBM stated, in the following quote, that it would upgrade Rational Application Developer and Software Architect to support WAS V6.1. From the WAS V6.1 announcement letter dated April 11, 2006:

> *“In 2006, both WebSphere and Rational plan to release major updates that are designed to be compatible with one another. WebSphere Application Server V6.1 is announced here. Rational is currently planning to ship Rational Application Developer and Rational Software Architect V7.0 in the second half of 2006. The gap between these product releases is intentional and is necessary to meet customer requirements for the WebSphere-specific functionality in Rational Application Developer and Rational Software Architect. During the timeframe between the availability of WebSphere Application Server V6.1 and Rational Application Developer V7.0 / Rational Software Architect V7.0, WebSphere customers who need to build and deploy applications to WebSphere Application Server V6.1 should use the Application Server Toolkit. Once V7.0 of the Rational products become available, those projects could be imported from the Application Server Toolkit to Rational Application Developer V7.0 / Rational Software Architect V7.0 to take advantage of the added capabilities*.”

IBM announced Rational Application Developer V7.0 and Rational Software Architect V6.0 on December 5, 2006, and the products were released on December 22, 2006.

Additional explore resources are available at the following Web sites:

- WAS V6.1 announcement letter:

  `http://www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=an&subtype=ca&appname=GPA&htmlfid=897/ENUS206-076`

- IBM developerWorks: Recommended reading list: J2EE and WebSphere Application Server:

http://www.ibm.com/developerworks/websphere/library/techarticles/0305_issw/recommendedreading.html

- Rational home page:

  http://www.ibm.com/software/rational

- IBM developerWorks Rational home page:

  http://www.ibm.com/developerworks/rational

- Rational Web Developer home page:

  http://www.ibm.com/software/awdtools/developer/web/index.html

- Rational Application Developer home page:

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

- Rational Software architect home page:

  http://www.ibm.com/software/awdtools/architect/swarchitect/index.html

4

Prepare the legacy application

In this chapter you configure the initial application database that will be used in the Experience J2EE! 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 like WAS V6.1 or AST V6.1. Instead, Derby only requires that the various jar and configuration files are copied onto the target system and available through the classpath configuration. This was done as part of the WAS V6.1 or AST V6.1 installations since they both ship with Derby; however, you could just as easily go to the Derby Web site and download the files.

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 developerWorks: Open source: Apache Derby project resources:

  `http://www.ibm.com/developerworks/opensource/top-projects/derby-community.html`

4.2 Create the Vacation application database

The Vacation application database simulates your legacy application. Since Derby does not have a graphical administration tool, you must create the database from the command line.

1. Ensure that you are logged on to the system as the user that will be used for most of the remainder of this book:
   - Windows User with local administrator privileges
   - Linux The user used to install the products in the previous chapter
2. Open a command line that is appropriate for your operating system.
3. Run the following commands:

– Windows

  ```
  C:
  CD \ExperienceJ2EE
  \IBM\AppServer\derby\bin\embedded\ij
  ```

  and at the ij command prompt execute:

  ```
  connect 'jdbc:derby:c:\ExperienceJ2EE\Vacation;create=true';
  run 'employee.ddl';
  exit;
  ```

– Linux

  ```
  cd $HOME/ExperienceJ2EE
  /opt/IBM/AppServer/derby/bin/embedded/ij.sh
  ```

  and at the ij command prompt execute:

  ```
  connect
  'jdbc:derby:/home/<user>/ExperienceJ2EE/Vacation;create=true';
  run 'employee.ddl';
  exit;
  ```

  where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable. However, you cannot use the $HOME variable with the ij command because it will not resolve the variable.

The following messages should be generated when running the DDL:

```
ij> CREATE TABLE "EXPERIENCEJ2EE"."EMPLOYEE"  (
        "EMPLOYEEID" INTEGER NOT NULL ,
        "FIRSTNAME" VARCHAR(40) NOT NULL ,
        "MIDDLENAME" VARCHAR(30) ,
        "LASTNAME" VARCHAR(40) NOT NULL ,
        "VACAMOUNT" INTEGER NOT NULL ,
        "SALARY" DECIMAL(7,0) );
0 rows inserted/updated/deleted
ij> ALTER TABLE "EXPERIENCEJ2EE"."EMPLOYEE"
        ADD CONSTRAINT "CC1086637357734" PRIMARY KEY
("EMPLOYEEID");
0 rows inserted/updated/deleted
```

Behind the scenes:

The employee.ddl file contains the commands that create the Vacation table and define the primary key:

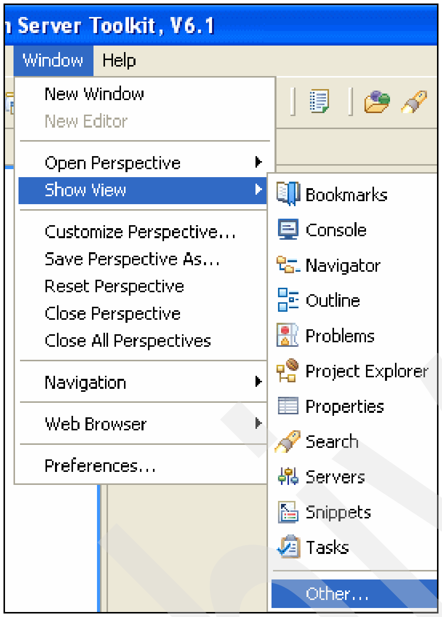
```
CREATE TABLE "EXPERIENCEJ2EE"."EMPLOYEE"  (
   “EMPLOYEEID" INTEGER NOT NULL ,
   "FIRSTNAME" VARCHAR(40) NOT NULL ,
   "MIDDLENAME" VARCHAR(30) ,
   "LASTNAME" VARCHAR(40) NOT NULL ,
   "VACAMOUNT" INTEGER NOT NULL ,
   "SALARY" DECIMAL(7,0) );
ALTER TABLE "EXPERIENCEJ2EE"."EMPLOYEE"
        ADD CONSTRAINT "CC1086637357734" PRIMARY KEY
("EMPLOYEEID");
```

4.3 Populate the Vacation application database

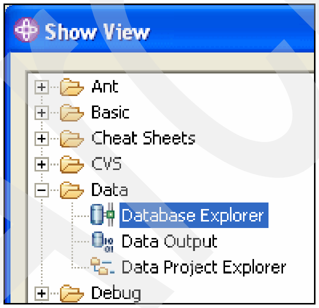
Next you add entries to the database. Since Derby does not have a graphical administration tool, you use some of the data tooling in AST V6.1 to add the entries.

4.3.1 Add the database to the Database Explorer view

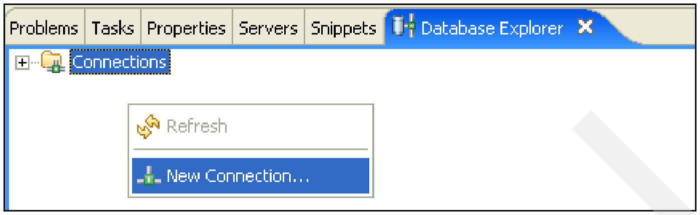
1. From the AST V6.1 **J2EE Perspective** action bar, select **Window** → **Show View** → **Other**.



2. From the **Show View** pop-up, select **Data** → **Database Explorer,** and click **OK**.



3. From the resulting **Database Explorer** view, in the lower-right pane), select **Connections**, and then right-click **New Connection...**.



4. At the **Connection Parameters** pop-up perform the following actions:

   – **Select a database manager**: Select **Derby** → **10.1**

   – **JDBC Driver**: **Derby Embedded JDBC Driver**

   – Windows **Database location**: **C:\ExperienceJ2EE\Vacation**

   – Linux **Database location**: **/home/*<user>*/ExperienceJ2EE/Vacation**

     where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.

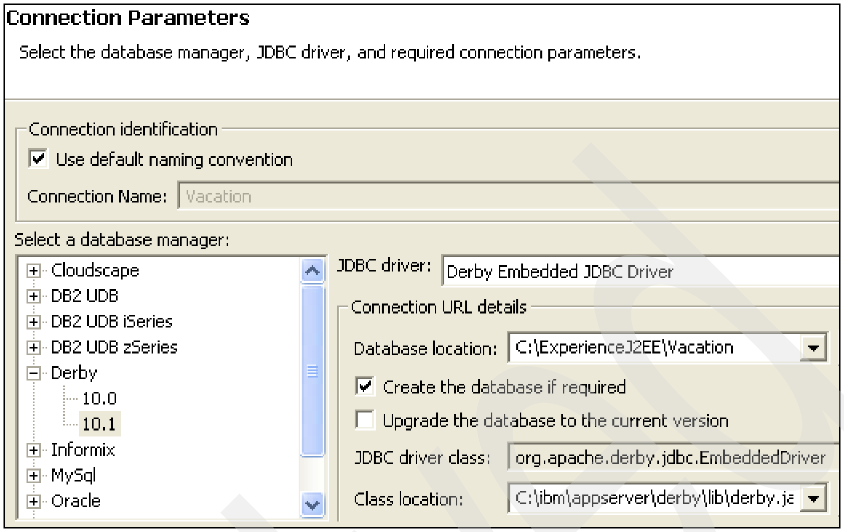
   – Windows **Class location**: **C:\IBM\AppServer\derby\lib\derby.jar**

   – Linux **Class location**: **/opt/IBM/AppServer/derby/lib/derby.jar**

   – Click **Test Connection**.

     If you entered the information properly, you will receive a pop-up stating **Connection to Derby is successful**. Click **OK** to close this **Test Connection** pop-up.

– Return to the **Connection Parameters** pop-up, and click **Finish**.



Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: Vacation

Select a database manager:

- Cloudscape
- DB2 UDB
- DB2 UDB iSeries
- DB2 UDB zSeries
- Derby
  - 10.0
  - 10.1
- Informix
- MySql
- Oracle

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\ExperienceJ2EE\Vacation

☑ Create the database if required

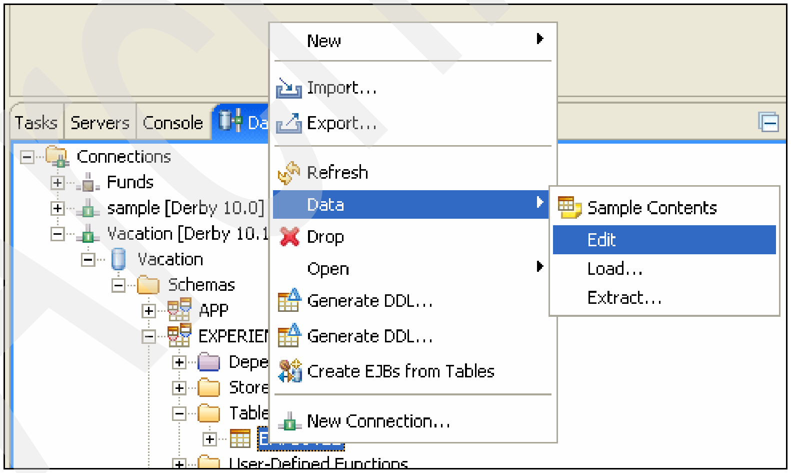
☐ Upgrade the database to the current version

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: C:\ibm\appserver\derby\lib\derby.ja

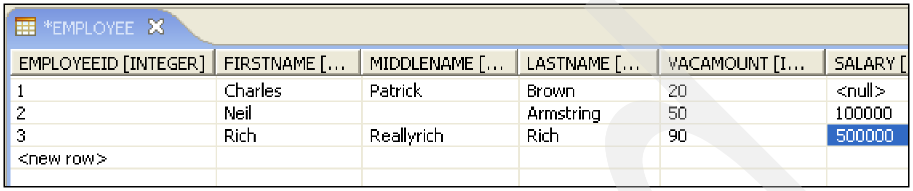
4.3.2 Edit the database

1. From the **Database Explorer** view, select **Connections** → **Vacation[Derby10.1]** → **Vacation** → **Schemas** → **EXPERIENCEJ2EE** → **Tables** → **EMPLOYEE**, and right-click **Data** → **Edit**.



2. At the resulting **EMPLOYEE** editor, in the upper-central pane, insert the three rows using the following actions:

- Under the **EMPLOYEEID[INTEGER]** column, click the value **<new row>**, and enter the employee ID (for example 1, 2 or 3), and press **Enter**.
- Repeat the previous step three times, once for each employee ID.
- Enter the information as follows for each employee. Some fields are intentionally left blank.

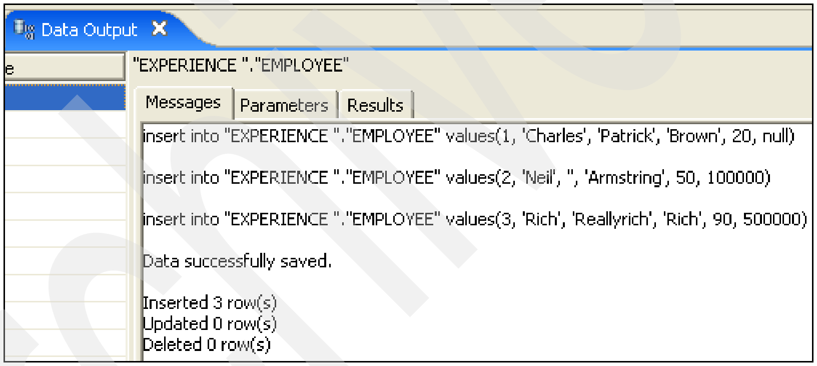


*EMPLOYEE ✕

EMPLOYEEID [INTEGER]	FIRSTNAME [...	MIDDLENAME [...	LASTNAME [...	VACAMOUNT [I...	SALARY [
1	Charles	Patrick	Brown	20	<null>
2	Neil		Armstring	50	100000
3	Rich	Reallyrich	Rich	90	500000
<new row>					

- Save the **EMPLOYEE** editor changes by clicking ☒ next to the name, and then clicking **Yes** at the **Save Resource** pop-up.

  Note the resulting **Data Output** view that appears in the lower right pane. This should indicate **Data successfully saved**.

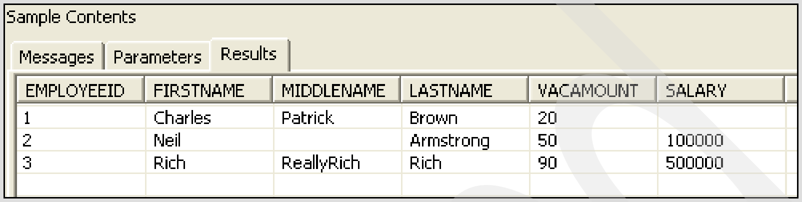


```
insert into "EXPERIENCE "."EMPLOYEE" values(1, 'Charles', 'Patrick', 'Brown', 20, null)

insert into "EXPERIENCE "."EMPLOYEE" values(2, 'Neil', '', 'Armstring', 50, 100000)

insert into "EXPERIENCE "."EMPLOYEE" values(3, 'Rich', 'Reallyrich', 'Rich', 90, 500000)

Data successfully saved.

Inserted 3 row(s)
Updated 0 row(s)
Deleted 0 row(s)
```

- Close this view by clicking ☒ next to the name.

Behind the scenes:

You can examine the inserted data by switching back to the Database Explorer view, selecting **Connections** → **Vacation[Derby10.1]** → **Vacation** → **Schemas** → **EXPERIENCEJ2EE** → **Tables** → **EMPLOYEE**, and right-clicking **Data** → **Sample Contents**.



EMPLOYEEID	FIRSTNAME	MIDDLENAME	LASTNAME	VACAMOUNT	SALARY
1	Charles	Patrick	Brown	20	
2	Neil		Armstrong	50	100000
3	Rich	ReallyRich	Rich	90	500000

4.4 Explore!

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.
- 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.

WAS V6.1/AST V6.1 contain both the Embedded and Network Server versions of Derby. This book, for simplicity of configuration, uses the Embedded version.

The primary disadvantage is that you cannot have concurrent access (for example by both AST V6.1 running in one JVM and WAS V6.1 running in another JVM). If you want to see if the entity EJB updated a database record, you must first shutdown WAS V6.1 before you can view the database with AST V6.1.

Additional explore resources are available at the following Web site:

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

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

Part 2

Core J2EE application

Part 2 describes how to implement a “traditional” J2EE application, starting with a single back-end relational database called Vacation, which we created in Chapter 4, “Prepare the legacy application” on page 71 and ending with a multi-tier application that implements the core J2EE elements of EJBs and JSPs, supporting both Web browsers and a thick Java client.

5

Create the employee data access element

In this chapter you create a new entity EJB that will coordinate and mediate access with the existing Vacation database in Derby in a *bottom up scenario*.

5.1 Learn!

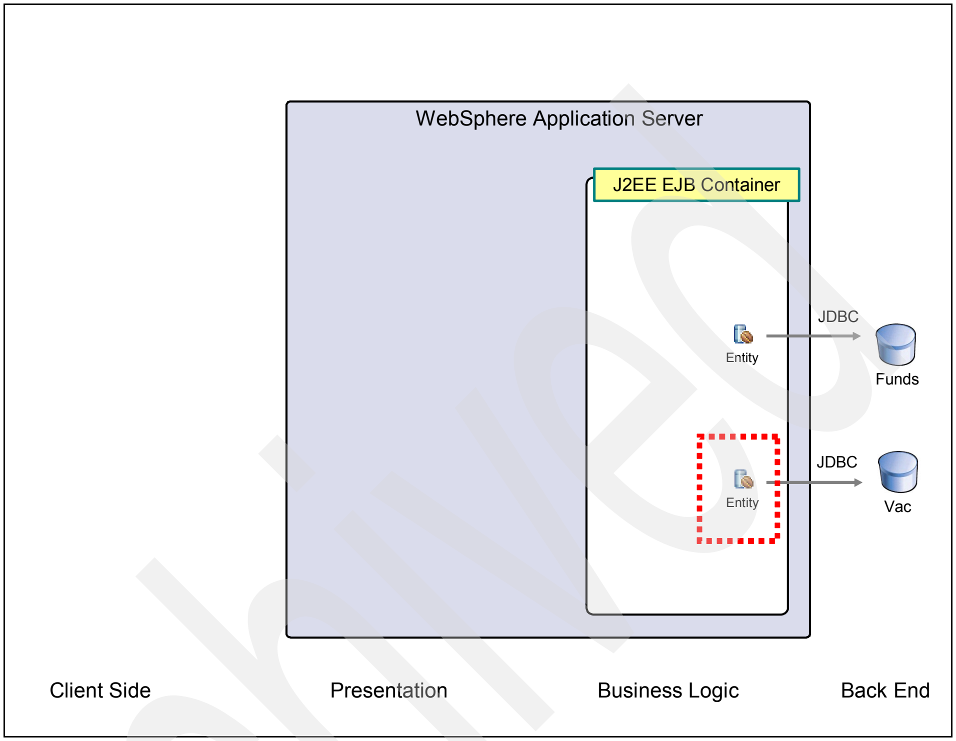


Figure 5-1 Donate application: Employee entity EJB

Enterprise JavaBeans (EJBs), as described earlier, represent the business logic of a J2EE application and run in a J2EE EJB container that provides runtime services. Following are the most critical services:

- Transactional coordination so the runtime handles the transactional starts, commits, and rollbacks across multiple applications.
- Local and remote invocation so user applications can transparently access EJBs that are on the same system or different systems. This also enables failover (if the EJB on one system is down you are redirected to another system) and workload management (the access of the EJBs are distributed equally across systems).
- Security constraints tied directly to the data or the business logic, providing a consistent security model across all access types, for example, through the Web container, thick clients, Web services, and messaging.

If you do not have any or all of these requirements (coordinated transactions, local/remote invocation, consistent security model), EJBs may be not be needed for your application.

J2EE contains the following three types of EJBs:

- Message-driven EJBs, which will be discussed in “Chapter 15. Create the message-driven bean” on page 349.
- Session EJBs, which will be discussed in “Chapter 7. Create the donate business logic element” on page 165.
- Entity EJBs, which are discussed in the following paragraph.

Entity EJBs provide encapsulated access to a record-oriented data element with a primary key, such as a row in a relational database.

The two primary types of entity EJBs are the following:

- Container managed persistence (CMP) entity EJBs, where the user data is kept in a relational database.

  The user defines the mapping between the CMP entity EJB fields and the back-end database table, and the J2EE tooling and infrastructure generates and executes all necessary code. The key emphasis here is that the user writes little or no code, and the J2EE infrastructure takes care of reading and writing the data.

- Bean managed persistence (BMP) entity EJB, where the user data is kept somewhere else.

  Similar to the CMP entity EJBs, the user defines the EJB fields, but then the user must write all the code required to read/write the records. The records could be in a flat file, in a XML file, or perhaps even in a relation database (that is not managed by J2EE).

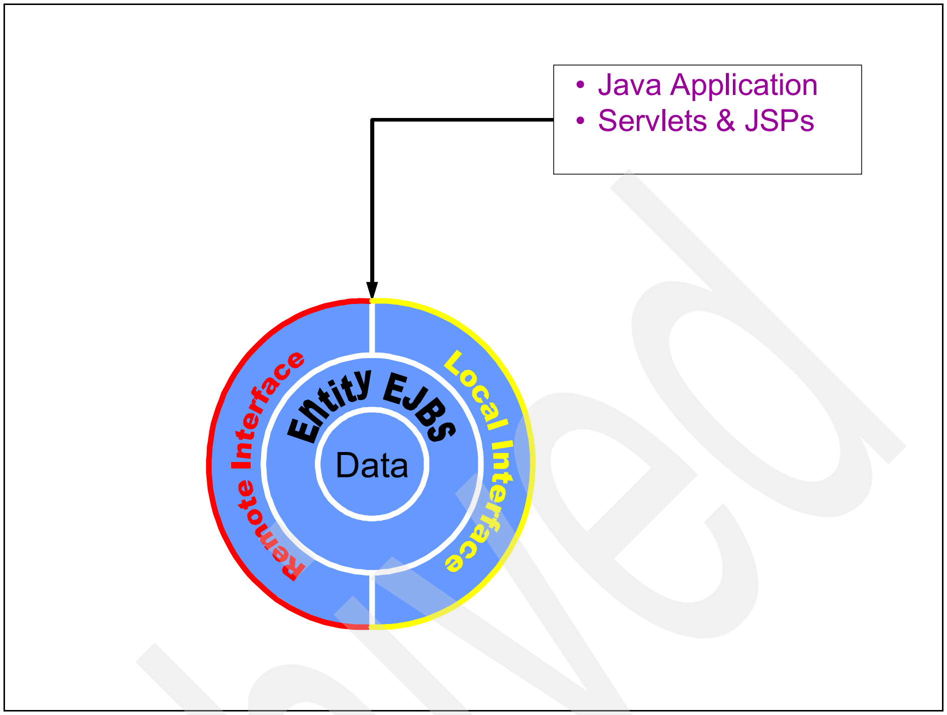


Figure 5-2 Entity EJBs

CMP entity EJBs are by far the most common. Consider Figure 5-2 as you read the following:

- Start with an existing database table.
- J2EE tooling supports the definition of an entity EJB based on this definition. This is called a *bottom-up* scenario. Most J2EE tooling also supports a *top-down* scenario (as you will do in “Chapter 6. Create the funds data access element” on page 135) and a *meet-in-the middle* scenario.

  This initial definition consists of the common deployment descriptors and Java classes that represent the “generic” form the of EJB that works across all J2EE runtime providers.

  – Entity EJBs are defined in a package artifact called an EJB project, and EJB projects are packaged together with other modules (such as other EJB projects, Java projects, Web projects, and Application Client projects) into an Enterprise Application project.

  An enterprise application is the artifact that is installed into a J2EE runtime.

- Prior to installing and running the entity EJB in a J2EE runtime, the developer or administrator must perform the following:

– Generate the deployment code that bridges between the generic EJB implementation and the specific database and J2EE runtime.

– Define a JNDI name (in the EJB deployment descriptor) that maps to the runtime database.

– Define the JDBC Provider and the JDBC data source in the runtime environment (with the same JNDI name previously used) that defines how to access the type and specific instance of the database.

> **Note:** This can be a different database than the database used to create the entity EJB through the *bottom-up scenario*, but it must contain the same table and schema definitions.

Once the entity EJB is installed and executing in a J2EE runtime, user programs can access the entity EJB. Entity EJBs can be accessed over two primary interfaces:

- The remote interface (shown in Figure 5-2 on page 86 in red) runs over remote method invocation (RMI) and allows any client to access the EJB over RMI. This interface supports transparent local and remote invocation along with the implied fail-over and workload management capabilities.
- The local interface (shown in Figure 5-2 on page 86 in yellow) was introduced in J2EE 1.3, and provides a ‘short-cut’ for invoking any EJB that is running in the same J2EE application server instance (meaning in the same JVM). This interface is more efficient because it is a direct Java to Java invocation, but it does not support fail-over or workload management.

You will use the local interface for your entity EJBs. We discuss this in further detail in Chapter 7, “Create the donate business logic element” on page 165.

The entity EJB can then be accessed through other Java programs:

- Local J2EE programming artifacts running in the same instance of the J2EE runtime can use the local interface for a high-performance invocation.
- Any Java program (within the same J2EE runtime instance or elsewhere) that is configured with the program jar files and configuration files can use the remote interface, accessing it through the RMI interface. The Java program needs to have local access to a subset of EJB project files. These files are typically packaged into an EJB Client jar file.

Following are the steps in a typical remote interface usage pattern:

- Issue a JNDI lookup call (a core J2EE service) to initialize the remote home Interface (initializing with the runtime RMI information that points to an EJB container that contains the entity EJB).

This call returns the entity EJB home interface, which effectively provides a factory that can be used to find, create, or remove the individual records represented by the entity EJB.

Alternative: The local interface does not use the standard JNDI lookup. Instead, you define an EJB reference in the deployment descriptor which provides the linkage to the appropriate Java classes. You will implement an EJB call using the local interface in section 7.3.2, “Add the call to the Employee entity EJB using a snippet” on page 181.

- Invoke the appropriate method of the home interface with an input primary key to return an entity EJB instance:
  - Create: Create a new entity EJB instance and a record in the associated database table.
  - Find: Locate an existing record in the associated database and populate the new entity EJB instance with the information from the existing record.

  The home interface also has a remove method that deletes a record from the database table.
- Work with the returned entity EJB instance to read or update the various fields in the record.

Additional learn resources are available at the following Web sites:

- EJB 2.1 Final Release:

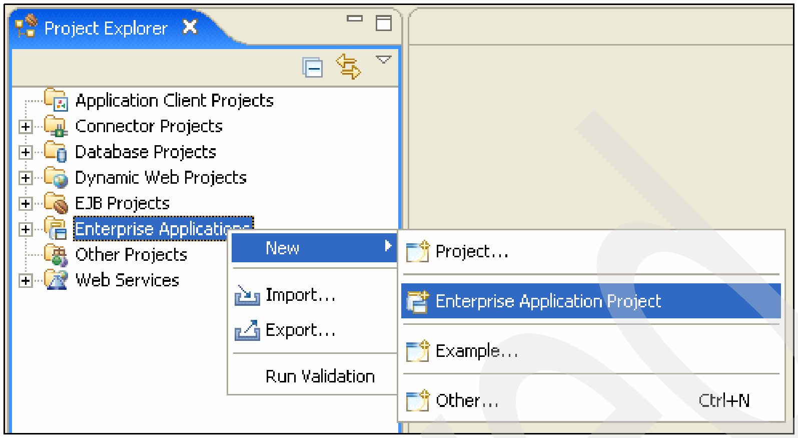
  `http://java.sun.com/products/ejb/docs.html`
- The J2EE 1.4 Tutorial: Chapter 23: Enterprise Beans:

  `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

5.2 Create the enterprise application project

In this section you create the DonateEAR enterprise application project and the DonateEJB EJB project that are used in this scenario.

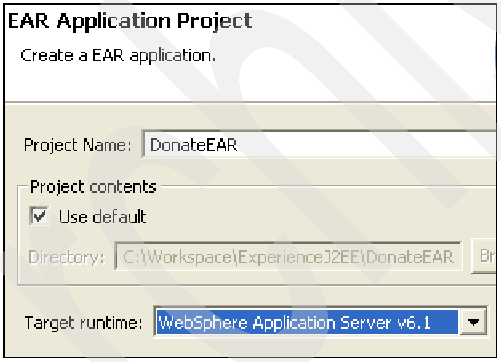
1. From the AST V6.1 **Project Explorer** view, select the **Enterprise Applications** entry, and right-click **New** → **Enterprise Application Project**.



2. At the **Enterprise Application Project** pop-up populate the following fields:

   – **Project Name**: **DonateEAR**

   – **Target Runtime**: **WebSphere Application Server v6.1**



EAR Application Project

Create a EAR application.

Project Name: DonateEAR

Project contents

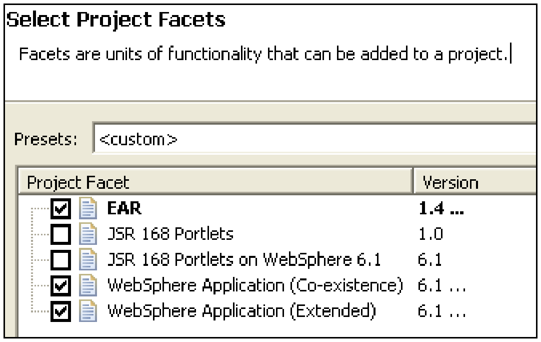
☑ Use default

Directory: C:\Workspace\ExperienceJ2EE\DonateEAR

Target runtime: WebSphere Application Server v6.1

   – Click **Next**.

3. At the **Select Project Facets** pop-up, click **Next**.



Select Project Facets

Facets are units of functionality that can be added to a project.

Presets: <custom>

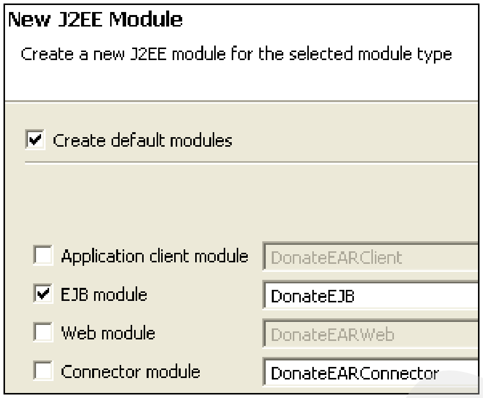
Project Facet	Version
☑ **EAR**	**1.4 ...**
□ JSR 168 Portlets	1.0
□ JSR 168 Portlets on WebSphere 6.1	6.1
☑ WebSphere Application (Co-existence)	6.1 ...
☑ WebSphere Application (Extended)	6.1 ...

Behind the scenes:

Project facets are a feature of WTP 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 WAS V6.1, supporting base J2EE capabilities (Co-existence) and IBM extensions (Extended).

You can change the list of project facets after a project is created by opening the project properties (select the project, and right-click **Properties**) and adding or removing facets on the **Project Facets** tab.

4. At the **J2EE Modules to Add to the EAR** pop-up, click **New Module…**
   – At the resulting **New J2EE Module** pop-up perform the following:
      - Clear **Application client module**, **Web module**, and **Connector module** options.
      - Rename the **EJB Module** to **DonateEJB**.
      - Click **Finish**.



New J2EE Module

Create a new J2EE module for the selected module type

☑ Create default modules

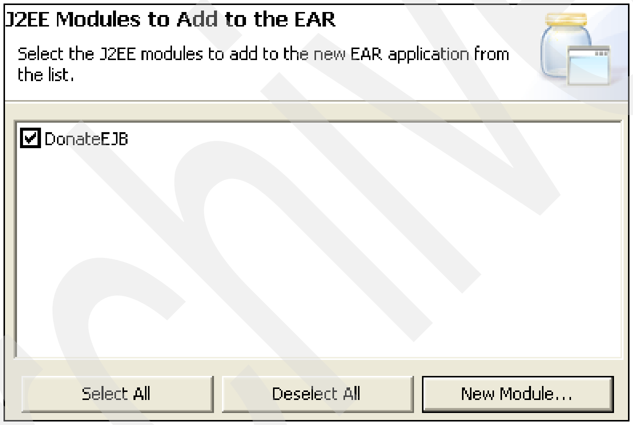
☐ Application client module: DonateEARClient

☑ EJB module: DonateEJB

☐ Web module: DonateEARWeb

☐ Connector module: DonateEARConnector

– Return to the **J2EE Modules to Add to the EAR** pop-up, ensure that **DonateEJB** is selected, and click **Finish**.



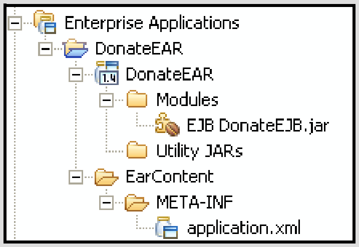
J2EE Modules to Add to the EAR

Select the J2EE modules to add to the new EAR application from the list.

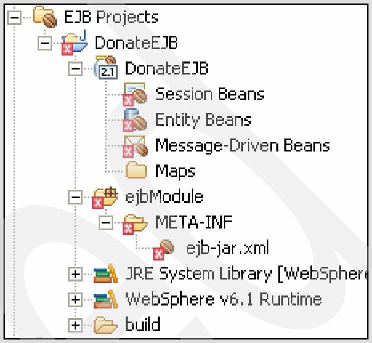
☑ DonateEJB

Behind the scenes:

This wizard created the base structure for the overall enterprise application project (**DonateEAR)** and for the EJB project (**Donate EJB**). Following is the expanded view of both of these (in the **Project Explorer** view):



- Enterprise Applications
  - DonateEAR
    - DonateEAR
      - Modules
        - EJB DonateEJB.jar
      - Utility JARs
    - EarContent
      - META-INF
        - application.xml



- EJB Projects
  - DonateEJB
    - DonateEJB
      - Session Beans
      - Entity Beans
      - Message-Driven Beans
      - Maps
    - ejbModule
      - META-INF
        - ejb-jar.xml
    - JRE System Library [WebSpher
    - WebSphere v6.1 Runtime
    - build

Note the red error notification symbols [☒] in the **DonateEJB** project. These indicate that an error or errors exist in the project. All errors and warnings are listed in the **Problems** view in the lower right pane, and in this case all of these error notification symbols are caused by a single error:

```
cvc-complex-type.2.4.b: The content of element 'ejb-jar' is not
complete. One of '{"http://java.sun.com/xml/ns/j2ee":display-name,
"http://java.sun.com/xml/ns/j2ee":icon,
"http://java.sun.com/xml/ns/j2ee":enterprise-beans}' is expected
```

This is a “normal” error, indicating that the EJB project contains no artifacts. From a J2EE perspective this is an error condition: An EJB project (jar) must contain at least one EJB bean, according to Chapter 23.1 Deployment Descriptor Overview of the EJB 2.1 specifications available at:

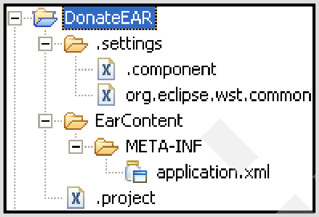
> http://java.sun.com/products/ejb/docs.html
>
> “*An ejb-jar file produced by the Bean Provider contains one or more enterprise beans...* “

You fix this error by creating an entity EJB in the next section.

Note that the project tree is an abstract view of the projects and does not represent the actual contained file based elements. The deployment descriptor subtrees (**DonateEAR** → **DonateEAR**, and **DonateEJB** → **DonateEJB**, for example) are a visual representation of the J2EE artifacts contained in each project, and as such represent one or more actual files in the actual project.

There are three common approaches to determine the actual file based elements:

- From the AST V6.1 application bar, select **Window** → **Show View** → **Navigator** to open the **Navigator** view (in the same pane as the **Project Explorer** view). This shows a file-based view of the project.

  Note that **Navigator** is also the default view in the **Resource** perspective.

- If you enable the Link option in the **Project Explorer** view action bar, the **Project Explorer** automatically expands to highlight the primary file based element when you open an abstract artifact like the **Enterprise Applications** → **DonateEAR** → **DonateEAR** deployment descriptor.

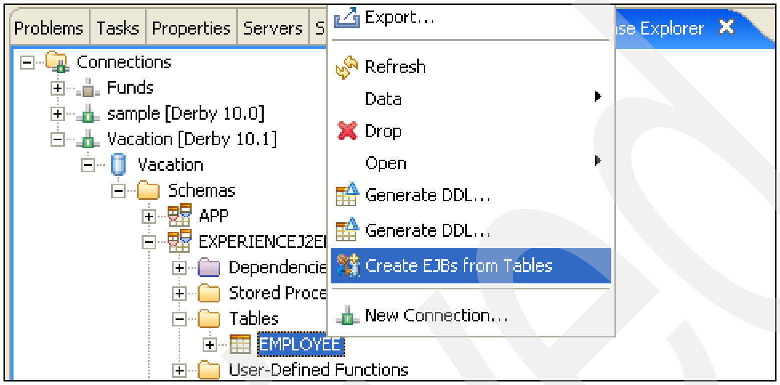
- Some editors (such as the Java editor) show the actual file resources when you hover over the name tab.

5.3 Develop the Employee entity EJB

In this section you use the existing Employee table definition (in the Vacation database) to create the Employee entity EJB.

5.3.1 Generate the entity EJB from the Vacation database

1. Switch to the **Database Explorer** view in the lower-right pane, and select **Connections** → **Vacation [Derby 10.1]** → **Vacation** → **Schemas** → **EXPERIENCEJ2EE** → **Tables** → **EMPLOYEE**. Right-click **Create EJBs from Tables**.



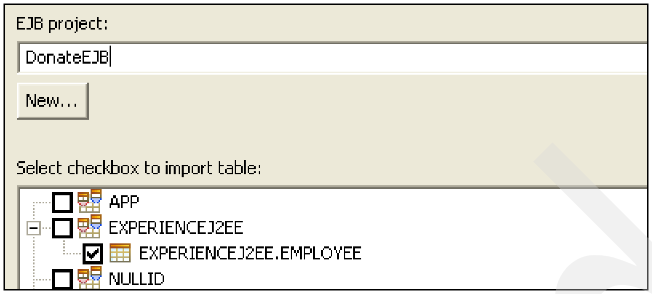
Off course?

- If the **Database Explorer** view does not appear, or if the **Vacation** database is not in the list of connections, repeat the steps in section “4.3.1 Add the database to the Database Explorer view” on page 75.
- If the **Vacation** connection cannot be expanded, then select it and right-click **Reconnect**. At the **Database Authorization** pop-up, click **OK** (no need to supply a user ID or password).

These issues may occur if you closed the view or restarted the workspace after following the steps in Chapter 4, “Prepare the legacy application” on page 71.

2. An **EJB to RDB Mapping** pop-up wizard appears. At the **Selective Database Import** pop-up perform the following:
   - In the EJB project field, type DonateEJB.
   - In the **Select check box** field, accept the default (**EXPERIENCEJ2EE.EMPLOYEE** is selected).

– Click **Next**.



EJB project:

DonateEJB

New...

Select checkbox to import table:

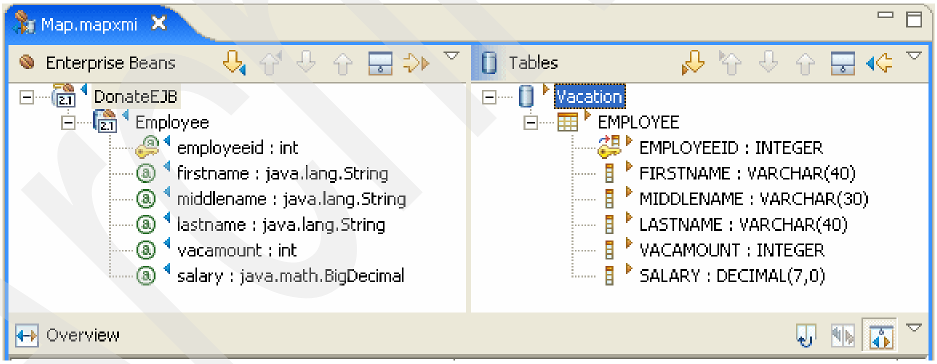
- □ APP
- □ EXPERIENCEJ2EE
  - ☑ EXPERIENCEJ2EE.EMPLOYEE
- □ NULLID

Work around:

The selection of the EJB project through the pull-down is not working in this release of AST V6.1, so you must manually type in DonateEJB.

– At the **Create new EJB/RDB Mapping** pop-up, click **Finish**.

3. The wizard opens a mapping file editor (**Map.mapxmi)** that describes the relationship between fields in the generated Employee entity EJB and the existing Employee table. View the mappings, and close the **Map.mapxmi** editor when done.



Map.mapxmi

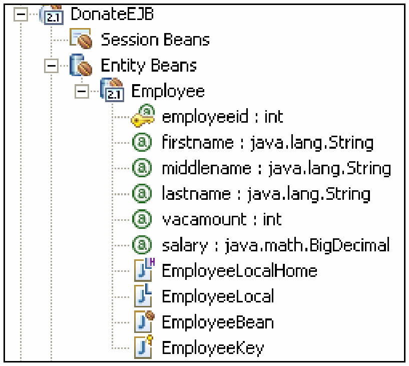
Enterprise Beans

- DonateEJB
  - Employee
    - employeeid : int
    - firstname : java.lang.String
    - middlename : java.lang.String
    - lastname : java.lang.String
    - vacamount : int
    - salary : java.math.BigDecimal

Tables

- Vacation
  - EMPLOYEE
    - EMPLOYEEID : INTEGER
    - FIRSTNAME : VARCHAR(40)
    - MIDDLENAME : VARCHAR(30)
    - LASTNAME : VARCHAR(40)
    - VACAMOUNT : INTEGER
    - SALARY : DECIMAL(7,0)

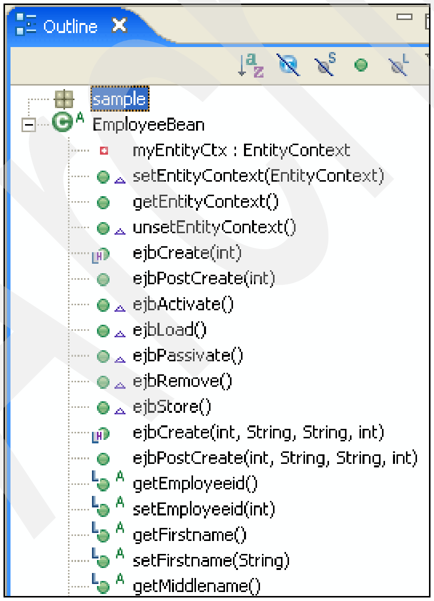
4. Expand to **EJB Projects** → **DonateEJB** → **DonateEJB** → **Entity Beans** → **Employee**. Note the entity EJB field definitions, which should match those shown in the mapping editor.



```
DonateEJB
  Session Beans
  Entity Beans
    Employee
      employeeid : int
      firstname : java.lang.String
      middlename : java.lang.String
      lastname : java.lang.String
      vacamount : int
      salary : java.math.BigDecimal
      EmployeeLocalHome
      EmployeeLocal
      EmployeeBean
      EmployeeKey
```

5. Double-click the Java artifact **EJB Projects** → **DonateEJB** → **DonateEJB** → **Entity Beans** → **Employee** → **EmployeeBean**(.java) to open in the Java editor.

6. Look at the **Outline** view in the upper right pane. Note the various getter and setter methods for the entity EJB data fields, and that they have a small L next to them—indicating that they are accessible through the EJB local interface. Close the **EmployeBean.java** editor when done.



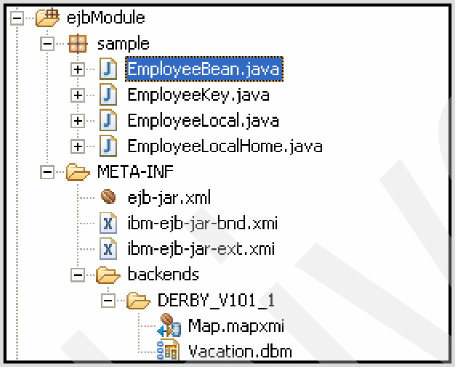
```
Outline
  sample
  EmployeeBean
    myEntityCtx : EntityContext
    setEntityContext(EntityContext)
    getEntityContext()
    unsetEntityContext()
    ejbCreate(int)
    ejbPostCreate(int)
    ejbActivate()
    ejbLoad()
    ejbPassivate()
    ejbRemove()
    ejbStore()
    ejbCreate(int, String, String, int)
    ejbPostCreate(int, String, String, int)
    getEmployeeid()
    setEmployeeid(int)
    getFirstname()
    setFirstname(String)
    getMiddlename()
```

Behind the scenes:

Note that you did not choose which interface to use, local or remote: The **Create EJBs from Tables wizard** creates an entity EJB with the local interface.

There are several ways to look at the J2EE artifacts created by this wizard.

- View the actual file artifacts in the workspace by expanding the **EJB Projects** → **DonateEJB** → **ejbModule subtree**.



```
ejbModule
  sample
    EmployeeBean.java
    EmployeeKey.java
    EmployeeLocal.java
    EmployeeLocalHome.java
  META-INF
    ejb-jar.xml
    ibm-ejb-jar-bnd.xmi
    ibm-ejb-jar-ext.xmi
    backends
      DERBY_V101_1
        Map.mapxmi
        Vacation.dbm
```

- View the EJB deployment descriptor by expanding to **EJB Projects** → **DonateEJB** → **DonateEJB** and double-clicking to open. The standard editor opens with a formatted view (with the last tab showing the source file).
- Graphically view the EJB deployment descriptor by expanding the entire **EJB Projects** → **DonateEJB** → **DonateEJB** subtree.

5.3.2 Disconnect from the Vacation database

The embedded version of Derby that you are using does not allow multiple programs to access a database concurrently. If you do not disconnect from the database (through the AST V6.1 tooling), subsequent access using the ExperienceJ2EE test server will fail because the database is already in use.

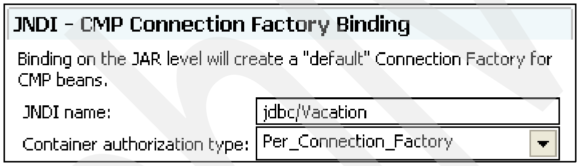
1. From the **Database Explorer** view, select **Connections** → **Vacation**.
2. Right-click **disconnect**.

5.3.3 Set the Default CMP connection factory JNDI name

The default CMP connection factory JNDI name (specified on the overview tab of the deployment descriptor) provides a single runtime mapping between all the entity EJBs in a project and a database instance. The default value is jdbc/Default, and you change this to a value that is unique for DonateEJB (jdbc/Vacation).

You could also choose to provide a unique CMP connection factory JNDI name for individual entity EJBs, specified through the bean tab of the deployment descriptor. This allows you to map individual entity EJBs to separate databases.

1. From the J2EE Perspective **Project Explorer**, select **EJB Projects** → **DonateEJB** → **DonateEJB**, and double-click to open the EJB deployment descriptor.
2. From the resulting **EJB Deployment Descriptor** editor (in the upper central pane), ensure that you are on the **Overview** tab.
3. Scroll down to **WebSphere Bindings**, and on the right under **JNDI - CMP Connection Factory Binding** change the **JNDI name** from **jdbc/Default** to **jdbc/Vacation**.



JNDI - CMP Connection Factory Binding

Binding on the JAR level will create a "default" Connection Factory for CMP beans.

JNDI name: jdbc/Vacation

Container authorization type: Per_Connection_Factory

4. Save and close the **EJB Deployment Descriptor**.

5.3.4 Generate the deployment code

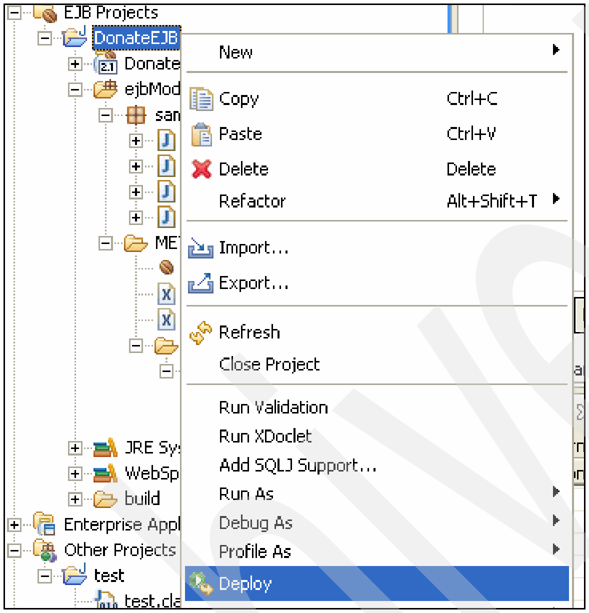
The deploy operation for entity EJBs creates the vendor and database specific implementation code needed to map between the abstract definition of the entity EJB and the actual runtime code.

You do not have to run this task now (during the development phase). Instead, you could wait and have the deployment code generated when installing the J2EE enterprise application (in this case DonateEAR) to the J2EE application server.

This is possible because the EJB deployment code can be generated either at development time (as you are doing now) or at installation time, when the EAR file (containing the EJB project/module) is installed into the J2EE runtime environment. In your case, this occurs when you publish DonateEAR in section 5.5, “Add the DonateEAR application to the test server” on page 114.

For the purpose of this scenario, you are generating the deployment code now before installing and publishing, so you can see what artifacts the deployment operation actually generates.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB**, and right-click **Deploy**.



The resulting pop-up automatically closes when the operation is completed.

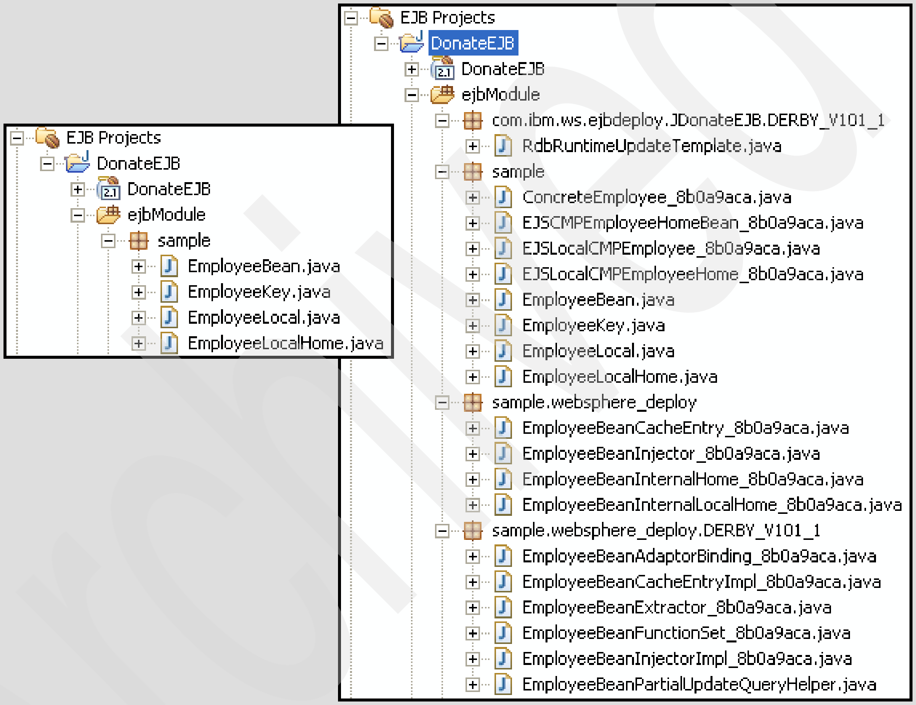
Behind the scenes:

The deploy operation creates the Java classes that are required to interface between the EJBs that you created and the runtime environment (in this case the WAS V6.1 test environment).

The classes are created under the **EJB Projects** → **DonateEJB** → **ejbModule** subtree:

Classes before deploy operation

Class after deploy operation

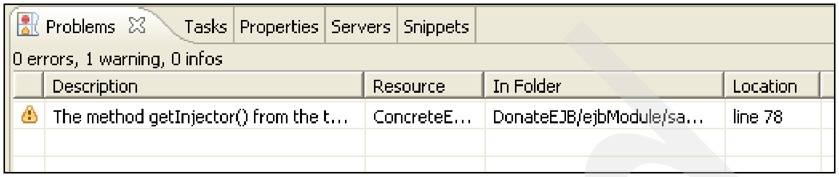


- EJB Projects
  - DonateEJB
    - DonateEJB
    - ejbModule
      - sample
        - EmployeeBean.java
        - EmployeeKey.java
        - EmployeeLocal.java
        - EmployeeLocalHome.java

- EJB Projects
  - DonateEJB
    - DonateEJB
    - ejbModule
      - com.ibm.ws.ejbdeploy.JDonateEJB.DERBY_V101_1
        - RdbRuntimeUpdateTemplate.java
      - sample
        - ConcreteEmployee_8b0a9aca.java
        - EJSCMPEmployeeHomeBean_8b0a9aca.java
        - EJSLocalCMPEmployee_8b0a9aca.java
        - EJSLocalCMPEmployeeHome_8b0a9aca.java
        - EmployeeBean.java
        - EmployeeKey.java
        - EmployeeLocal.java
        - EmployeeLocalHome.java
      - sample.websphere_deploy
        - EmployeeBeanCacheEntry_8b0a9aca.java
        - EmployeeBeanInjector_8b0a9aca.java
        - EmployeeBeanInternalHome_8b0a9aca.java
        - EmployeeBeanInternalLocalHome_8b0a9aca.java
      - sample.websphere_deploy.DERBY_V101_1
        - EmployeeBeanAdaptorBinding_8b0a9aca.java
        - EmployeeBeanCacheEntryImpl_8b0a9aca.java
        - EmployeeBeanExtractor_8b0a9aca.java
        - EmployeeBeanFunctionSet_8b0a9aca.java
        - EmployeeBeanInjectorImpl_8b0a9aca.java
        - EmployeeBeanPartialUpdateQueryHelper.java

5.3.5 Alter the workspace settings to ignore unused methods

Warnings will appear in the **Problems** view (in the lower right pane) after generating the EJB deploy code. All of these warnings should indicate that the Java files contain methods that are never used.

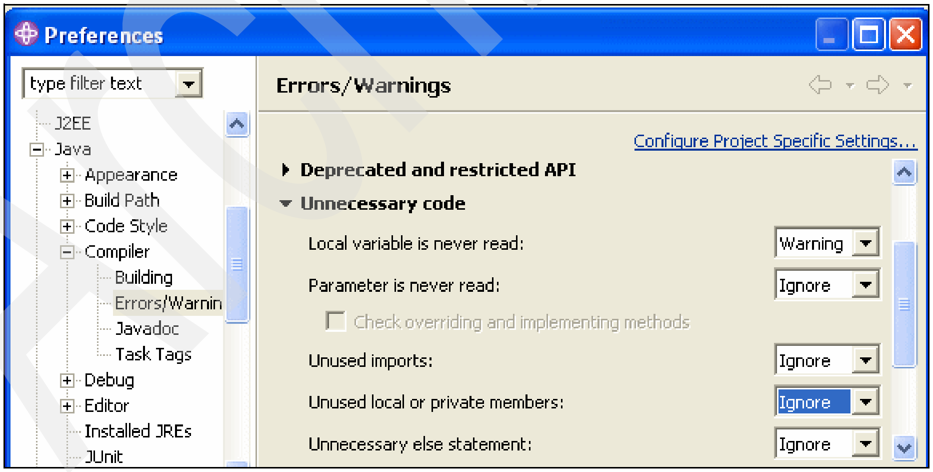


0 errors, 1 warning, 0 infos

	Description	Resource	In Folder	Location
⚠	The method getInjector() from the t...	ConcreteE...	DonateEJB/ejbModule/sa...	line 78

You can ignore these warnings because the unneeded method might take up a small amount of space, but will not impact the runtime behavior.

1. From the AST V6.1 **J2EE Perspective** action bar, select **Window** → **Preferences**.
2. At the **Preferences** pop-up perform the following:
   - – On the left, select **Java** → **Compiler** → **Errors/Warnings**.
   - – On the right (at the resulting **Errors/Warnings** pane), expand the **Unnecessary code** section, and change the option for **Unused local or private members** from **Warning** to **Ignore**.
   - – Click **OK**.
   - – At the resulting **Error/Warnings Settings Changed** pop-up, select **Yes**.



Local variable is never read: Warning

Parameter is never read: Ignore

□ Check overriding and implementing methods

Unused imports: Ignore

Unused local or private members: Ignore

Unnecessary else statement: Ignore

3. The **Problems** view should now show no errors or warnings.

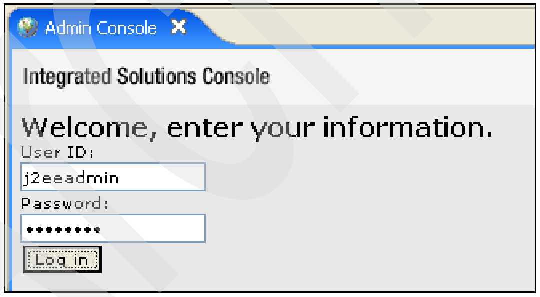
5.4 Configure the test server for DonateEJB

The ExperienceJ2EE Server @ localhost test server, which will serve as your J2EE runtime environment, needs to be configured with the proper JDBC provider and JDBC data source to map from the DonateEJB project to the Vacation database.

5.4.1 Start the admin console

The WAS V6.1 Administration 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 **Servers** view in the lower-right pane, and verify that the **ExperienceJ2EE Server @ localhost** is already started—that it has a status of started. If it is not, select it, and click the start icon [▶] in the application action bar to start it.
2. Select the **ExperienceJ2EE Server @ localhost**, and right-click **Run administrative console**. Perform the following tasks.
   – At the browser prompt for the certificate, accept the certificate.
   – At the **Admin Console** login screen, set **User id** to **j2eeadmin** and **Password** to the value you set in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51.
   – Click **Log in**.



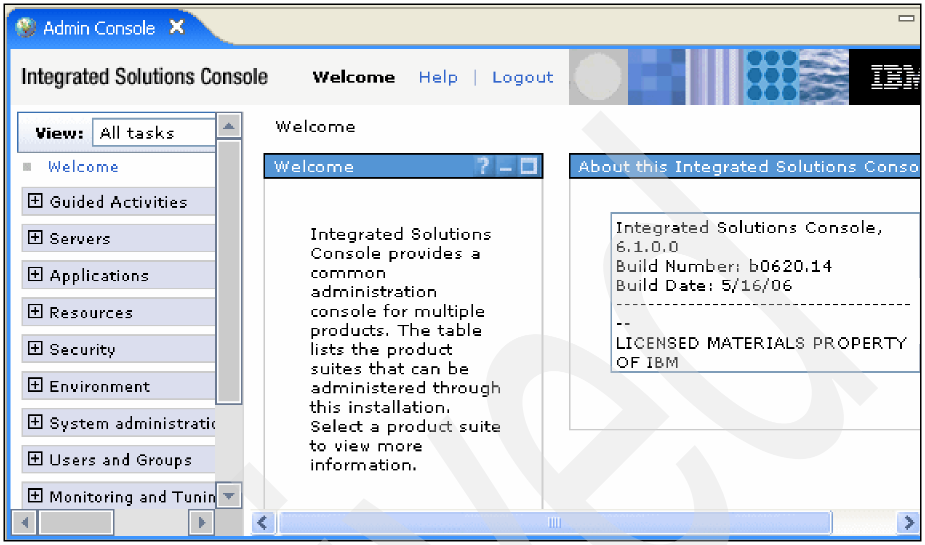
Integrated Solutions Console

Welcome, enter your information.

User ID: j2eeadmin

Password: ••••••••

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



Integrated Solutions Console provides a common administration console for multiple products. The table lists the product suites that can be administered through this installation. Select a product suite to view more information.

Integrated Solutions Console, 6.1.0.0
Build Number: b0620.14
Build Date: 5/16/06

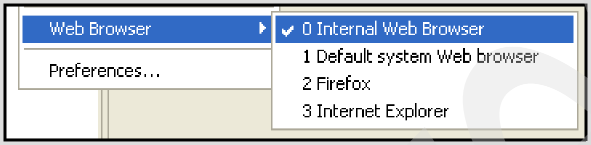
--
LICENSED MATERIALS PROPERTY OF IBM

Off course?

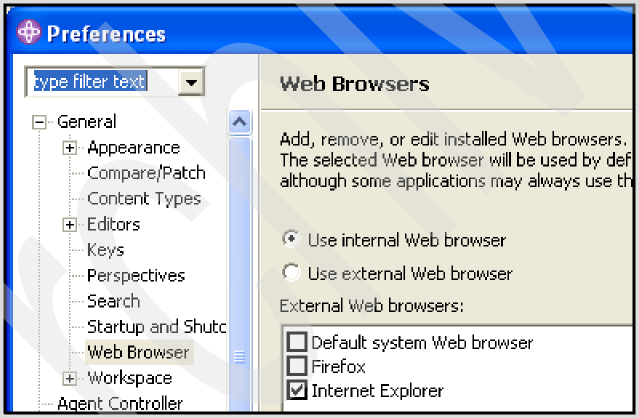
If the Admin Console appears in an external Web browser, your AST V6.1 workspace was not configured to use the internal Web browser (previously shown).

AST V6.1 and Eclipse allow you to specify which browser to use. Following are two ways of specifying this:

- From the application action bar, select **Window** → **Web Browser**, and then select the desired value.



- From the application action bar, select **Windows** → **Preferences**. Then, from the resulting **Preferences** pop-up, expand to **General** → **Web Browser**, and select the desired value.



Web Browsers

Add, remove, or edit installed Web browsers.
The selected Web browser will be used by def
although some applications may always use th

● Use internal Web browser

○ Use external Web browser

External Web browsers:

☐ Default system Web browser
☐ Firefox
☑ Internet Explorer

Linux If the **Use internal Web browser** option is grayed out (not selectable), your browser configuration does not meet the AST V6.1/Eclipse requirements. See section 2.4.2, “Linux operating system considerations” on page 28 for more details.

5.4.2 Create the ExperienceJ2EE JAAS authentication alias

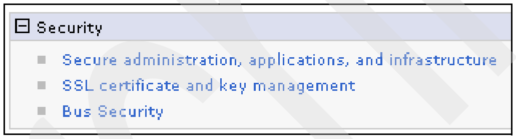
The Java authentication and authorization service (JAAS) authentication alias provides the runtime user identity that is used when a J2EE artifact, such as an EJB, accesses a J2EE resource such as a JDBC data source.

You might reasonably ask why do you need a JAAS authentication alias if you are originally running without J2EE application security (and connecting to a database manager that does not use security)?

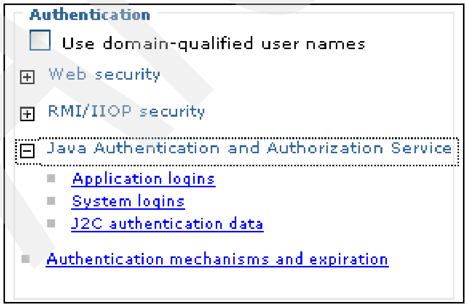
The answer to the question is because you have administrative security enabled. A side effect of this is that XA recovery actions need a user ID and password. Otherwise, you will receive the following error messages in the **Console** view when an XA recovery is attempted (and re-attempted every minute thereafter):

```
[7/7/06 12:43:40:968 CDT] 0000001d XARecoveryDat W   WTRN0005W: The
XAResource for a transaction participant could not be recreated and
transaction recovery may not be able to complete properly. The
resource was J2CXAResourceInfo :
cfName = Vacation Data source_CF
configProps = [Deployed Resource Adapter Properties]
   XA_RECOVERY_AUTH_ALIAS  java.lang.String  (none)
```

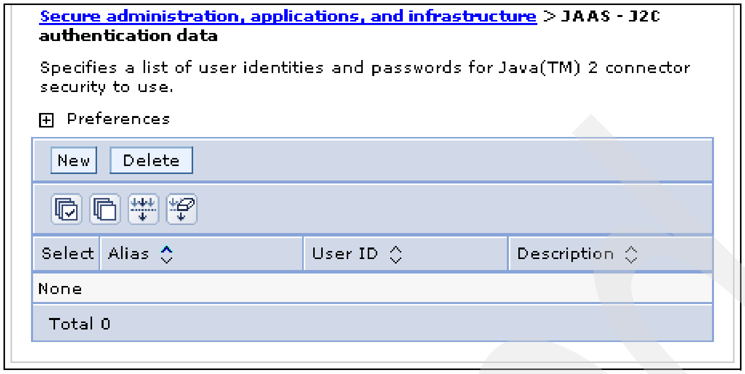
1. From the **Admin Console**, on the left expand to **Security**, and click **Secure administration, applications, and infrastructure**.


⊟ Security
Secure administration, applications, and infrastructure
SSL certificate and key management
Bus Security

2. On the right (in **Secure administration, applications, and infrastructure**), under **Authentication**, expand **Java authentication and Authorization Service**. Select **J2C authentication data**.


Authentication
□ Use domain-qualified user names
⊞ Web security
⊞ RMI/IIOP security
⊟ Java Authentication and Authorization Service
Application logins
System logins
J2C authentication data
Authentication mechanisms and expiration

3. On the right, in **Secure administration, applications, and infrastructure** → **JAAS** → **J2C authentication data**, click **New**.



Secure administration, applications, and infrastructure > JAAS - J2C authentication data

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

⊞ Preferences

Select	Alias ⇕	User ID ⇕	Description ⇕
None			
Total 0			

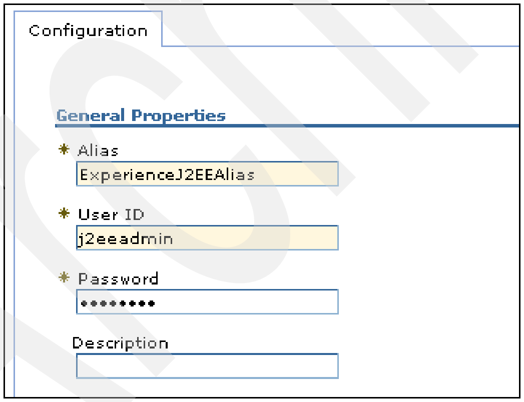
4. On the right, in **Secure administration, applications, and infrastructure** → **JAAS** → **J2C authentication data** → **New** perform the following:

   – **Alias**: **ExperienceJ2EEAlias**

   – **User ID**: **j2eeadmin**

   – **Password**: Password created in section “3.5 Create the ExperienceJ2EE application server profile” on page 51.

   – Click **Apply**.



General Properties

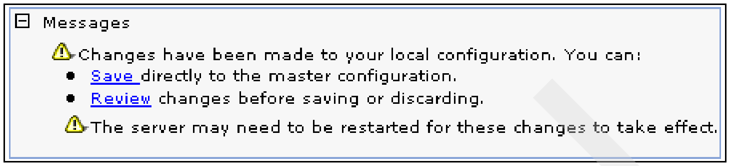
* Alias
ExperienceJ2EEAlias

* User ID
j2eeadmin

* Password
••••••••

Description

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.



⊟ Messages

⚠ Changes have been made to your local configuration. You can:

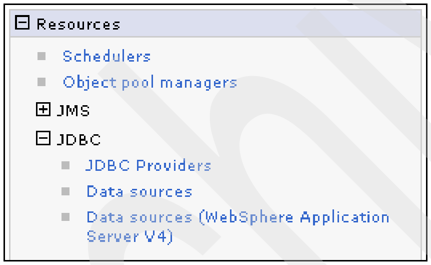
- Save directly to the master configuration.
- Review changes before saving or discarding.

⚠ The server may need to be restarted for these changes to take effect.

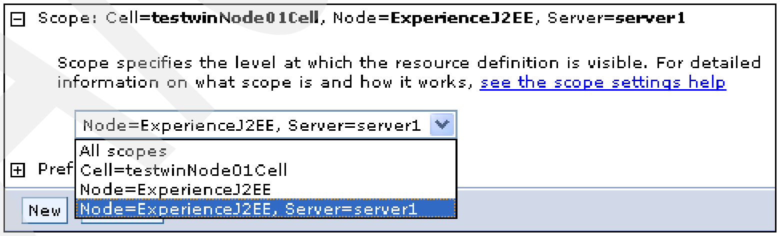
5.4.3 Create the Derby XA JDBC provider

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. From the **Admin Console**, on the left expand to **Resources** → **JDBC** → **JDBC Providers**.



2. On the right, at the **JDBC providers** panel, under **Scope** use the pull-down to change the scope from **All scopes** to **Node=ExperienceJ2EE, Server=server1**.



⊟ Scope: Cell=**testwinNode01Cell**, Node=**ExperienceJ2EE**, Server=**server1**

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help

Node=ExperienceJ2EE, Server=server1
All scopes
Cell=testwinNode01Cell
Node=ExperienceJ2EE
Node=ExperienceJ2EE, Server=server1
⊞ Pref
New

Behind the scenes:

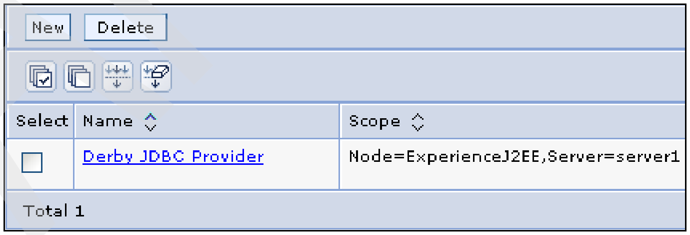
WAS V6.1 with the Deployment Manager component supports the administration of a group of systems, each running one or more instances of WAS V6.1.

The file-based configuration (that is, the profile you created in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51) 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 testj2ee and this is the second profile, the cell name is testj2eeNode02Cell.
- 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 testj2ee, the default value for the first node in the cell would be testj2eeNode01. You overrode that value when creating the profile and instead used ExperienceJ2EE.
- 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 ExperienceJ2EE 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.

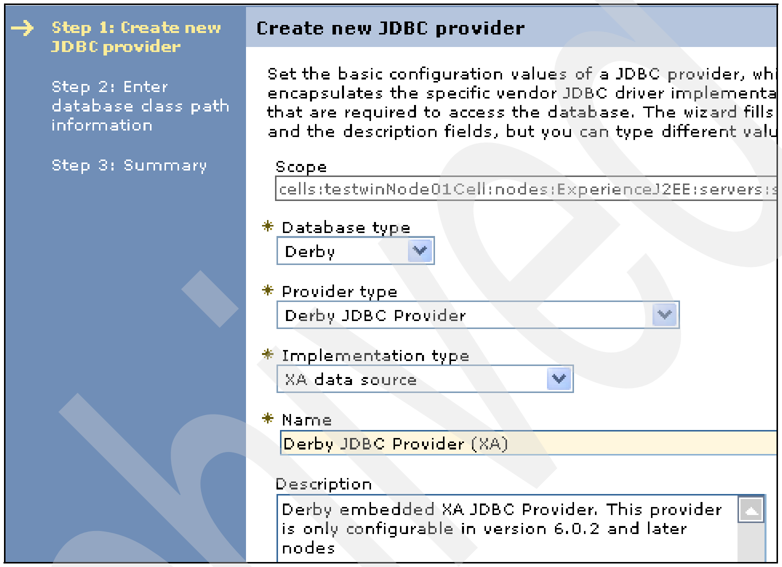
3. Click **New** to launch the JDBC provider wizard.



Select	Name	Scope
□	Derby JDBC Provider	Node=ExperienceJ2EE,Server=server1

Total 1

4. At the **Step 1: Create new JDBC Provider** panel perform the following:
   - **Database type**: **Derby**
   - **Provider type**: **Derby JDBC Provider**
   - **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, whi
encapsulates the specific vendor JDBC driver implementa
that are required to access the database. The wizard fills
and the description fields, but you can type different valu

Scope
cells:testwinNode01Cell:nodes:ExperienceJ2EE:servers:s

* Database type
Derby

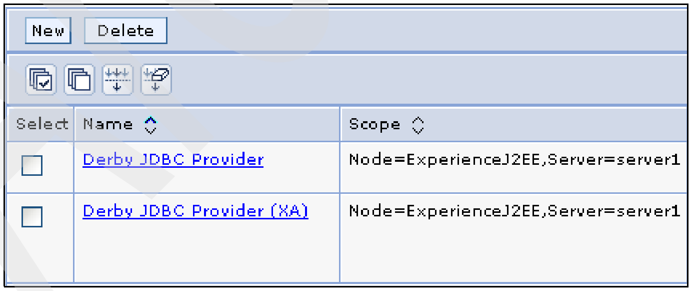
* Provider type
Derby JDBC Provider

* Implementation type
XA data source

* Name
Derby JDBC Provider (XA)

Description
Derby embedded XA JDBC Provider. This provider is only configurable in version 6.0.2 and later nodes

5. At the **Step 3: Summary** panel, click **Finish**. Note that we skipped Step 2 because the Derby classpath is already configured.
6. Return to the **JDBC providers** panel, and note the new entry.

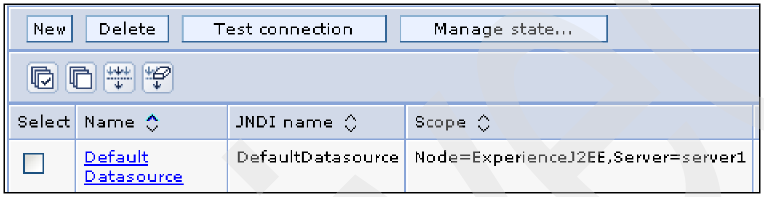


Select	Name	Scope
□	Derby JDBC Provider	Node=ExperienceJ2EE,Server=server1
□	Derby JDBC Provider (XA)	Node=ExperienceJ2EE,Server=server1

5.4.4 Create the Vacation JDBC data source

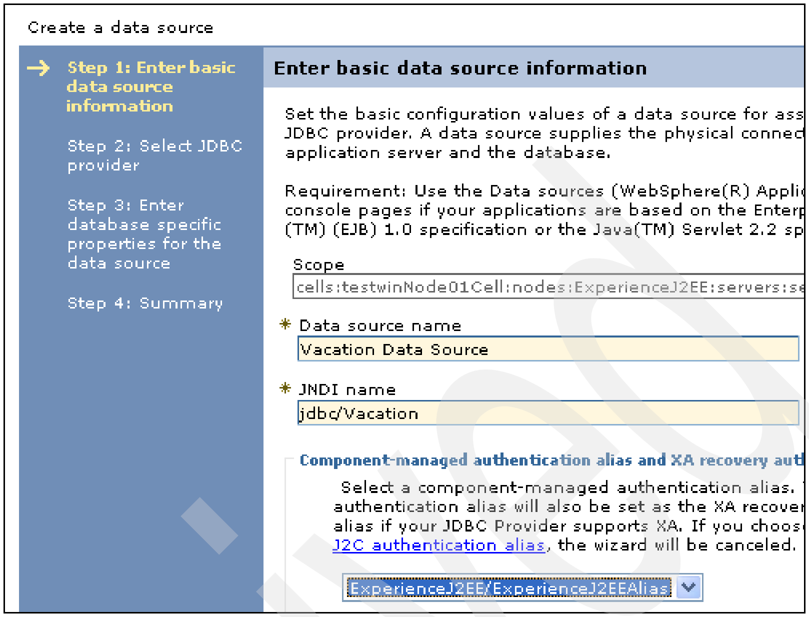
The JDBC data source defines the connection to the specific database (in this case the Vacation database created in section 4.2, “Create the Vacation application database” on page 72).

1. From the **Admin Console**, on the left expand to **Resources** → **JDBC** → **Data sources**.
2. On the right at the **Data sources** panel, under **Scope** use the pull-down to change the scope from **All scopes** to **Node=ExperienceJ2EE, Server=server1**.
3. Click **New** to launch the Data sources wizard.



Select	Name	JNDI name	Scope
□	Default Datasource	DefaultDatasource	Node=ExperienceJ2EE,Server=server1

4. At the **Step 1: Enter basic data source information** panel perform the following:
   – **Data source name**: **Vacation Data source**
   – **JNDI name**: **jdbc/Vacation**
   – Under **Component-managed authentication alias and XA recovery authentication alias**, select **ExperienceJ2EE/ExperienceJ2EEAlias** from the pull-down.
   – Click **Next**.



Create a data source

→ Step 1: Enter basic data source information
Step 2: Select JDBC provider
Step 3: Enter database specific properties for the data source
Step 4: Summary

Enter basic data source information

Set the basic configuration values of a data source for ass JDBC provider. A data source supplies the physical connec application server and the database.

Requirement: Use the Data sources (WebSphere(R) Appli console pages if your applications are based on the Enterp (TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 sp

Scope
cells:testwinNode01Cell:nodes:ExperienceJ2EE:servers:se

* Data source name
Vacation Data Source

* JNDI name
jdbc/Vacation

Component-managed authentication alias and XA recovery aut

Select a component-managed authentication alias. authentication alias will also be set as the XA recover alias if your JDBC Provider supports XA. If you choos J2C authentication alias, the wizard will be canceled.

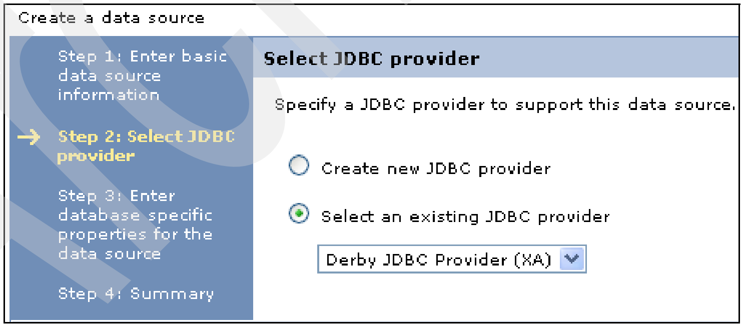
ExperienceJ2EE/ExperienceJ2EEAlias

5. At the **Step 2: Select JDBC provider** panel perform the following:

– Select **Select an existing JDBC provider**.

– Select **Derby JDBC Provider (XA)** from the pull-down list.

– Click **Next**.



Create a data source

Step 1: Enter basic data source information
→ Step 2: Select JDBC provider
Step 3: Enter database specific properties for the data source
Step 4: Summary

Select JDBC provider

Specify a JDBC provider to support this data source.

○ Create new JDBC provider

● Select an existing JDBC provider

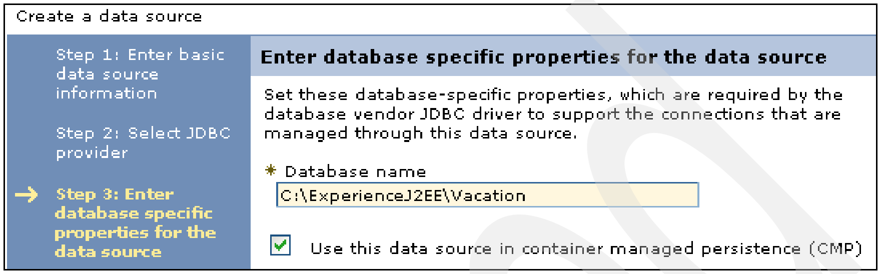
Derby JDBC Provider (XA)

6. At the **Step 3: Enter database specific properties for the data source** panel perform the following:

– Windows **Database Name**: **C:\ExperienceJ2EE\Vacation**

– Linux **Database Name**: **/home/*<user>*/ExperienceJ2EE/Vacation** where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.

- Select **Use this data source in container managed persistence (CMP)**.
- Click **Next**.


Create a data source
Step 1: Enter basic data source information
Step 2: Select JDBC provider
→ Step 3: Enter database specific properties for the data source

Enter database specific properties for the data source

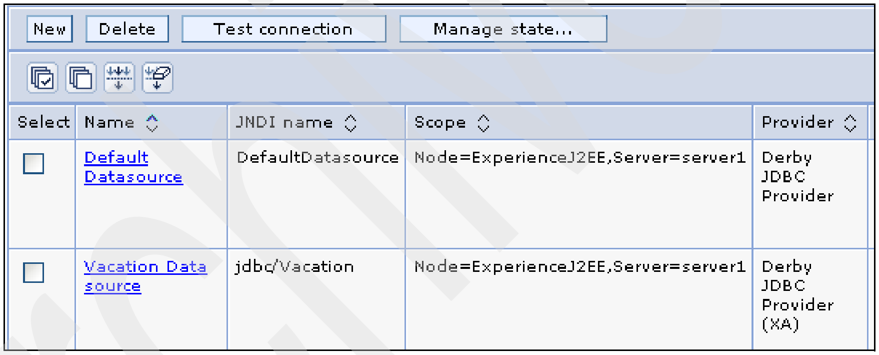
Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through this data source.

* Database name: C:\ExperienceJ2EE\Vacation

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

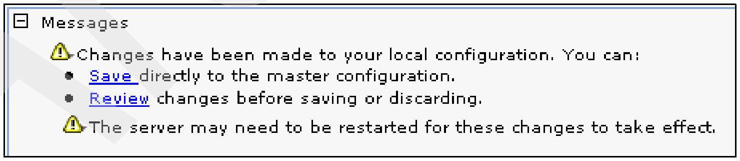
7. At the **Step 4: Summary panel**, click **Finish**.

8. Return to the **Data sources** panel, and note the new entry.



Select	Name	JNDI name	Scope	Provider
□	Default Datasource	DefaultDatasource	Node=ExperienceJ2EE,Server=server1	Derby JDBC Provider
□	Vacation Data source	jdbc/Vacation	Node=ExperienceJ2EE,Server=server1	Derby JDBC Provider (XA)

9. At the top of the panel, in the **Messages** box that states "**Changes have been made to your local configuration….**", click **Save**.



⚠ Changes have been made to your local configuration. You can:

- Save directly to the master configuration.
- Review changes before saving or discarding.

⚠ The server may need to be restarted for these changes to take effect.

Behind the scenes:

Changes made through the Admin Console are not active until you save the changes and they are written out to the ExperienceJ2EE 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.

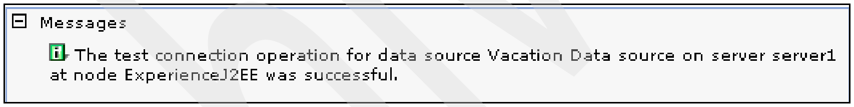
In this case, you have made all the necessary changes, and in fact you must save before testing the data source that you just created.

10. Test the JDBC configuration:

   – Return to the **Data sources** window.

   – Select the check box next to **Vacation** data source.

   – Click **Test connection**. The test connection should show the following message:



⊟ Messages

ℹ The test connection operation for data source Vacation Data source on server server1 at node ExperienceJ2EE was successful.

Off course?

If the test connection fails with the following message:

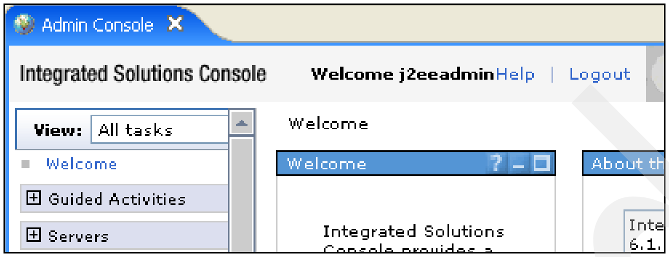
```
The test connection operation failed for data source Vacation
Data source on server server1 at node ExperienceJ2EE with the
following exception: java.sql.SQLException: Failed to start
database 'C:\ExperienceJ2EE\Vacation', see the next exception
for details.DSRA0010E: SQL State = XJ040, Error Code = 40,000.
View JVM logs for further details.
```

It means you neglected to disconnect from the Vacation database connection in the **Database Explorer** view, as described in section 5.3.2, “Disconnect from the Vacation database” on page 97.

The embedded version of Derby that you are using only allows a single program to access a database at a time. Since AST V6.1 was accessing the database through the Database Explorer, this access (using the WAS V6.1 test server) failed.

11. Click **Logout** located at the top of the **Admin Console**.

12. After the log out completes, close the **Admin Console**.



Behind the scenes:

If you do not log out before closing the **Admin Console**, the browser session within AST V6.1 will have a JSESSIONID that maps to an authentication token with WAS V6.1. This token will time out within 30 minutes.

If you attempt to re-open the **Admin Console** after 30 minutes, WAS V6.1may attempt to re-use this token, and the access will fail with the following message in the **Console** view:

```
SECJ0371W: Validation of the LTPA token failed because the token
expired with the following info: Token expiration Date: Thu Sep
21 11:35:08 CDT 2006, current Date: Thu Sep 21 11:58:08 CDT 2006.
```

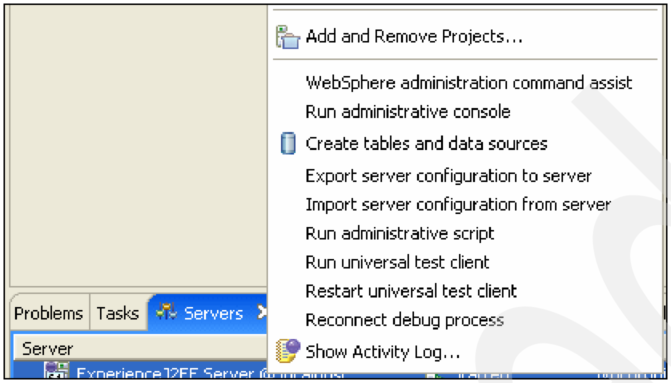
The solution to this problem is to stop and restart both the **ExperienceJ2EE Server @localhost** test server and the overall AST V6.1 IDE.

5.5 Add the DonateEAR application to the test server

In a normal production environment, at this point you would export the enterprise application project from AST V6.1 as an enterprise archive file (EAR). This EAR file would contain all the child projects (in this case DonateEJB) and could be installed into the J2EE application server.

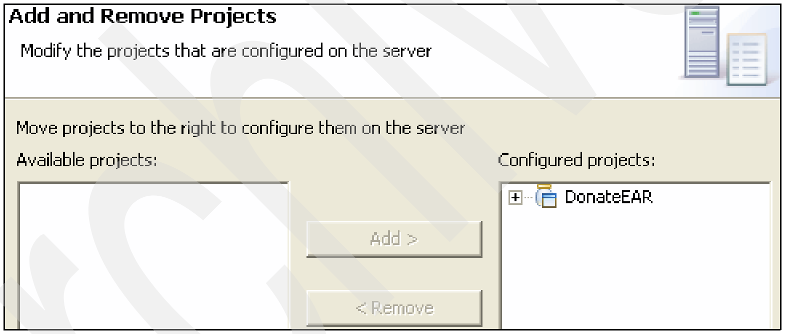
However, since you are in a development environment, AST V6.1 allows you to maintain the application in the workspace, and instead simply add the enterprise application project to the test server configuration. AST V6.1 automatically installs the application and, as you make further changes to the application in subsequent chapters, automatically publishes the changes to the server.

1. From the **J2EE Perspective**, switch to the **Servers** view in the lower-right pane. Select **ExperienceJ2EE Server @ localhost**, and right-click → **Add and remove projects….**


Add and Remove Projects…
WebSphere administration command assist
Run administrative console
Create tables and data sources
Export server configuration to server
Import server configuration from server
Run administrative script
Run universal test client
Restart universal test client
Reconnect debug process
Show Activity Log…
Problems
Tasks
Servers
Server
ExperienceJ2EE Server @ localhost

2. At the **Add and Remove Projects** pop-up, select **DonateEAR** in the left pane, and click **Add>** to move it to the right pane.

3. Click **Finish**.


Add and Remove Projects

Modify the projects that are configured on the server

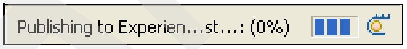
Move projects to the right to configure them on the server

Available projects:

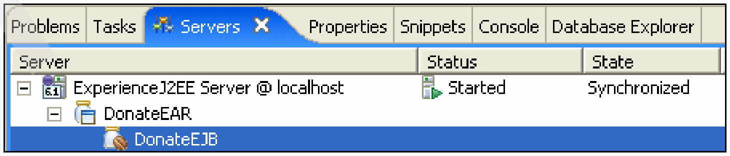
Configured projects:

DonateEAR

4. The deployment will take place in the background, with the status displayed in the lower right task bar. Wait for the task to complete.


Publishing to Experien…st…: (0%)

Note that the **Servers** view in the lower right is updated to include the deployed application:



Server	Status	State
ExperienceJ2EE Server @ localhost	Started	Synchronized
DonateEAR		
DonateEJB		

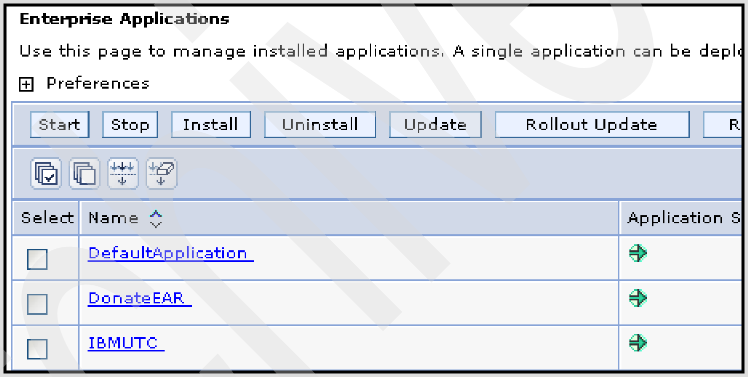
5. Switch to the **Console** view in the lower-right pane, and verify that the application was added to the configuration and started without errors. You may have to scroll up to see these relevant messages:

```
WSVR0200I: Starting application: DonateEAR
WSVR0204I: Application: DonateEAR  Application build level: Unknown
WSVR0037I: Starting EJB jar: DonateEJB.jar
WSVR0057I: EJB jar started: DonateEJB.jar
WSVR0221I: Application started: DonateEAR
```

   Note: In the Console view, each message is prefixed with a status header similar to [6/20/06 13:50:44:153 CDT] 00000014 ApplicationMg A.

Behind the scenes:

The enterprise application was added to the server configuration and automatically started. You can view the installed application using the Admin Console (select **Applications** → **Enterprise Applications**):



Enterprise Applications

Use this page to manage installed applications. A single application can be depl

⊞ Preferences

Select	Name ⇕	Application S
□	DefaultApplication	➜
□	DonateEAR	➜
□	IBMUTC	➜

The **Servers** view also has the option to automatically create the required tables and data sources (select the server and right-click **Create tables and data sources**). This can be done as an alternative to manually creating the JDBC provider and data sources as done in section 5.4, “Configure the test server for DonateEJB” on page 102.

- The Create Tables and Datasources wizard creates definitions for the JDBC providers/data sources in a section of the EAR called the Enhanced EAR support (an IBM J2EE extension).

The Enhanced EAR support allows the developer to specify required configurations, such as JDBC resources, within the EAR file. These are automatically created in the Application Server when the EAR is installed. This information can be found in the EAR deployment descriptor on the Deployment tab.

The Enhanced EAR definitions, if they exist, can also be viewed in the META-INF ibmconfig subdirectory in the EAR project.

Why did you not use the **Create tables and data sources** wizard?

- Historical: You could not use this wizard with previous J2EE tooling packages because the earlier versions of this wizard were somewhat limited in function.

  For example, the wizard shipped in Rational Application Developer V6.0 only created one-phase commit JDBC providers, and you require two-phase commit (XA) JDBC providers for this scenario.

  However, with AST V6.1 you actually COULD use the wizard if you desired, because the wizard shipped with AST V6.1 has significant enhancements that allow you to (1) specify two-phase commit (XA) JDBC providers and (2) use existing databases/tables.

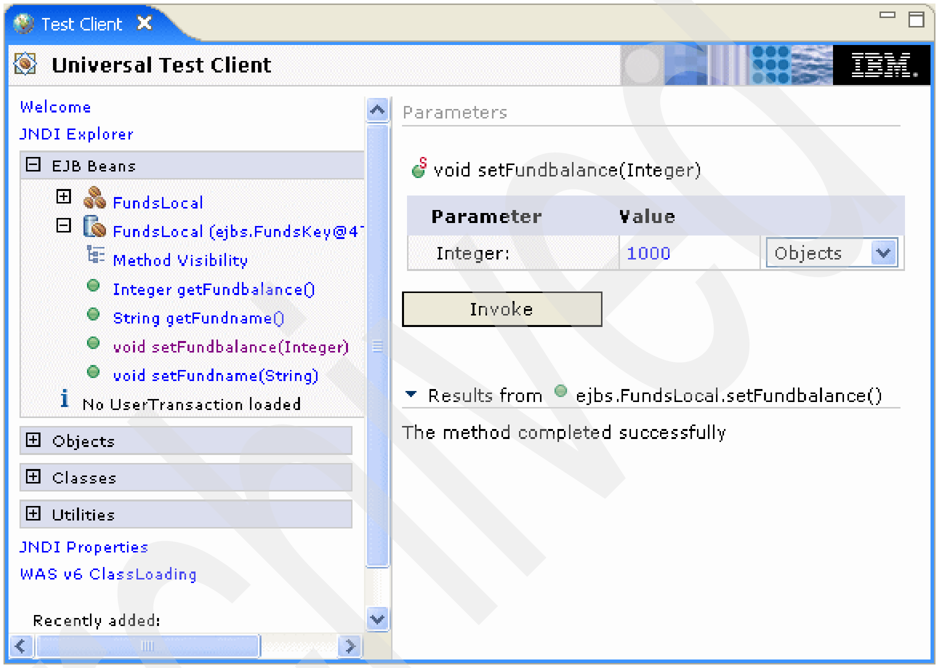
- Portability: The Enhanced EAR file-based artifacts are an IBM WebSphere extension. Therefore, if you used this option these scenarios are less portable to other J2EE application servers/development environment.

- Common user experience: The representation/user interface used to configure this information in AST V6.1 (using the wizard or the Deployment tab in the EAR deployment descriptor) is different than the representation/user interface used in the WAS V6.1 Administration Console (as you did in section “5.4 Configure the test server for DonateEJB” on page 102).

  One could argue that the AST V6.1 representation is simpler than the WAS V6.1 representation. However, is this enough to justify the introduction of a separate approach, thus complicating the transition from configuring this information in the development environment to the production or test environment? The author’s opinion was no, but this is a question where others may reasonably arrive at a different answer.

5.6 Test the Employee entity EJB with the Universal Test Client

The Universal Test Client (UTC) is a test tool shipped with AST V6.1 that allows you to perform basic testing of your J2EE artifacts without writing test code. For example, in this chapter you use the UTC to test entity EJBs using the local interface.



Parameter	Value	
Integer:	1000	Objects

The method completed successfully

- 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 WAS v6 ClassLoading 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 result on the right. For the image above, 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 WAS v6 ClassLoading) you can set properties such as the user ID and password

Behind the scenes:

J2EE documentation typically describes the name space as being a single entity, but in reality it is a combination of the following three separate name spaces, Server, Local, and Java:

- Server: contains the objects that are accessible by remote processes, as well as by local processes. One server name space exists for the entire server. In the UTC screens that will follow, the server name space is all the entries AFTER Local. Thus, the Employee entity EJB is NOT listed in the server name space because it is only accessible through the local interface.

  The Donate session EJB (which you create in Chapter 5, “Create the employee data access element” on page 83) will have a remote interface, and will be accessible through the server name space under **ejb** → **sample2** → **DonateHome**).

  WAS V6.1 contains a **`dumpNameSpace`** command that can dump this entry from the server name space:

  ```
  (top)/nodes/ExperienceJ2EE/servers/server1/ejb/sample2/DonateHome
          sample2.DonateHome
  ```

  For example, the following code sample would locate this EJB using standard J2EE JNDI lookup code (instead of the IBM specific ServiceLocatorManager):

  ```
  InitialContext initCtx = new InitialContext();
  DonateHome home = (DonateHome)
       initCtx.lookup("ejb/sample2/DonateHome");
  ```

- Local: contains objects that are available by local processes only. One local name space exists for the entire server. The Employee entity EJB created above has a local interface and is thus available through the local interface. In the UTC screens that follow, the Employee entity EJB is clearly marked as being in the local name space because it appears under Local.

  The local name space cannot be displayed using the **`dumpNameSpace`** command. Instead, you must use the NameServer MBean command to perform a dumpLocalNameSpace operation (see the InfoCenter reference in section 5.7, “Explore!” on page 131).

  For example, the following code sample would locate this EJB using standard J2EE JNDI lookup code:

  ```
  EmployeeLocalHome home = (EmployeeLocalHome)
       testI.lookup("local:ejb/ejb/sample/EmployeeLocalHome");
  ```

- Java: contains naming information unique to the specific server application. By default, when an application queries the java name space it receives the binding information for itself only and not for other applications on the server.

  The java name space cannot be displayed using the **dumpNameSpace** command. Instead, you must use the NameServer MBean command to perform a dumpJavaNameSpace operation (see the InfoCenter reference in section 5.7, “Explore!” on page 131).

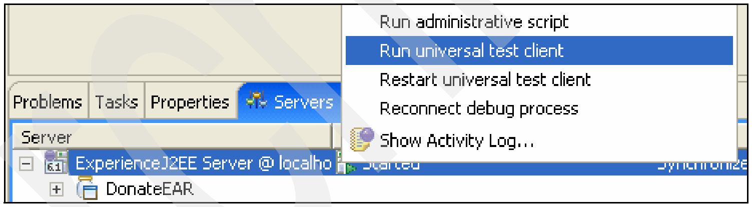
  For example, the following code sample would locate this EJB using standard J2EE JNDI lookup code:

  ```
  EmployeeLocalHome home = (EmployeeLocalHome)
  testI.lookup("java:comp/env/ejb/Employee");
  ```

  This code will map through the EJB reference (that will be defined in Chapter 7, “Create the donate business logic element” on page 165) to the local name.

5.6.1 Start the UTC

1. Switch to the **Servers** view in the lower-right pane.
2. Select the **ExperienceJ2EE Server @ localhost**, and right-click **Run universal test client**.


Problems
Tasks
Properties
Servers
Server
ExperienceJ2EE Server @ localho
Started
Synchronize
DonateEAR
Run administrative script
Run universal test client
Restart universal test client
Reconnect debug process
Show Activity Log...

The UTC will open as a browser window in the upper-right pane called **Test Client**. Note that you can also access this test client through an external Web browser using the following URL:

http://localhost:9080/UTC

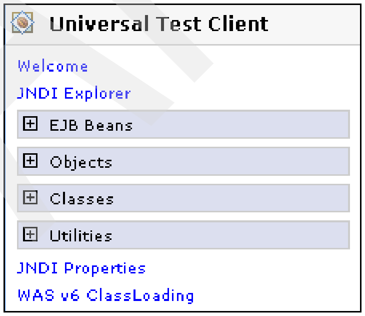
Behind the scenes:

If you upgrade AST V6.1 *after* creating a test server definition within AST V6.1, you may have to follow these steps to upgrade the UTC to the latest version:

1. Ensure that the test server is started.
2. Open the test server definition (within the **Servers** view of AST V6.1).
3. Switch to the **Overview** tab, and clear **Enable universal test client**.
4. Save and close the test server definition.
5. Switch to the **Console** view, and verify that the IBMUTC.EAR is stopped or removed.
6. Stop AST V6.1.
7. Start AST V6.1 (note that you can leave the test server running).
8. Start the Test Server within AST V6.1, and verify that messages appear in the **Console** view. Note that since the Test Server is already started, this action will cause AST V6.1 to reattach.
9. Open the test server definition (with the **Servers** view of AST V6.1).
10. Switch to the **Overview** tab, and select the **Enable universal test client** option.
11. Save and close the test server definition.
12. Switch to the **Console** view, and verify that the IBMUTC.EAR is added and started.

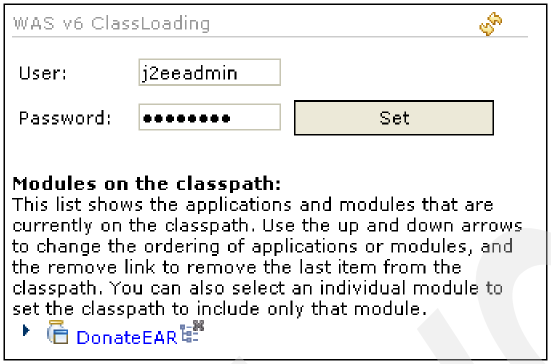
5.6.2 Specify the Admin user ID

1. From the **Test Client** in the upper left, select **WAS v6 ClassLoading**.



2. From the **WAS v6 ClassLoading** pane (in the right), set **User** to **j2eeadmin** and **Password** to the value you set in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51.

3. Click **Set**. Note that the selection under Modules on the classpath should change from ○ No applications found to ▸ DonateEAR .



WAS v6 ClassLoading

User: j2eeadmin

Password: ●●●●●●●●●

Modules on the classpath:
This list shows the applications and modules that are currently on the classpath. Use the up and down arrows to change the ordering of applications or modules, and the remove link to remove the last item from the classpath. You can also select an individual module to set the classpath to include only that module.

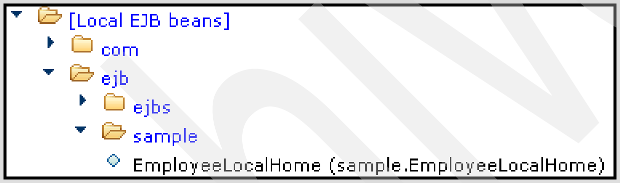
▸ DonateEAR

Behind the scenes:

The UTC will not function properly when the WAS V6.1 administrative security is enabled unless you configure the user ID and password for the classloader access. By default The UTC attempts to view the WAS V6.1 classpath information as UNAUTHENTICATED, and, since the J2EE administration security is enabled, the lookup fails with the following message in the **Console** view:

```
SECJ0305I: The role-based authorization check failed for
admin-authz operation
AppManagement:listApplications:java.util.Hashtable:java.lang.Stri
ng.  The user UNAUTHENTICATED (unique ID: unauthenticated) was
not granted any of the following required roles:
adminsecuritymanager, administrator, monitor, configurator,
operator, deployer.
```

As result, the Local EJB Beans entries in the UTC JNDI explorer are grayed out (not selectable):



```
[Local EJB beans]
  com
  ejb
    ejbs
    sample
      EmployeeLocalHome (sample.EmployeeLocalHome)
```

One work around is to test the entity EJB in the UTC:

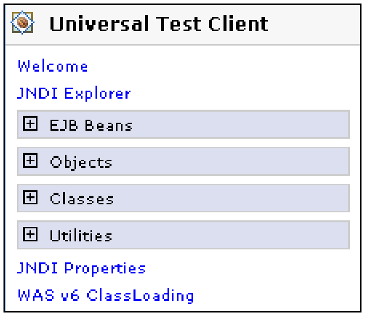
- Select the entity EJB in the Project Explorer (for example, **EJB Projects** → **DonateEJB** → **DonateEJB** → **Entity EJBs** → **Employee**).
- Right-click **Run As** → **Run on Server**. This does not eliminate the JNDI error described above, but it does allow you to test the entity EJB.

Another work around is to set the user ID and password that the UTC uses to interrogate the classpath.

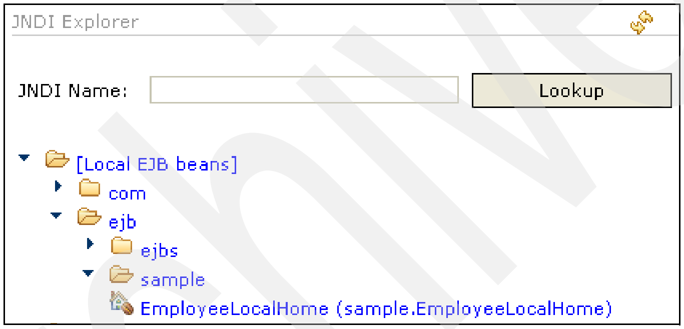
Note that this user ID/password setting is not preserved across restarts of the UTC; therefore, you must enter these each time you restart the UTC.

5.6.3 Test the Employee entity EJB with the UTC

1. From the **Test Client** in the upper left, select **JNDI Explorer**.



2. From the **JNDI Explorer** pane, locate the reference to the entity EJB local home interface by expanding to **[Local EJB Beans]** → **ejb** → **sample** and left-clicking on **EmployeeLocalHome**.



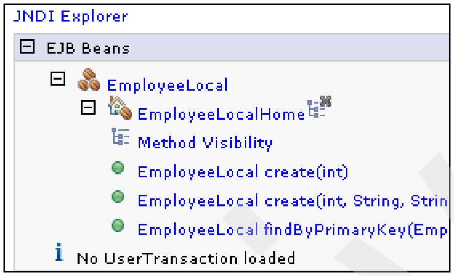
Off course?

If the EmployeeLocalHome reference is grayed out in the above pane (and therefore not selectable) or not visible at all, then the UTC experienced errors in interacting with the name space. In that case, you can launch the EJBs into the UTC from the **Project Explorer**:

- Select **EJB Projects** → **DonateEJB** → **DonateEJB** → **Entity Beans** → **employee,** and right-click **Run As** → **Run on Server**.
- At the resulting **Define a New Server** pop-up, select **ExperienceJ2EE Server @ localhost**, and click **Finish**.

> **Behind the scenes**: Local EJBs are not really JNDI resources and are thus listed in a special section, [Local EJB beans], in the JNDI namespace. If you were testing EJBs with remote interfaces, you would look for the EJB reference in the formal JNDI namespace (as you will do in section 7.4, "Test the Donate session EJB with the UTC" on page 187).

3. From the **EJB Beans** section located on the left, locate the desired entity EJB instance by expanding to **EJB Beans** → **EmployeeLocal** → **EmployeeLocalHome**. Click **findbyPrimaryKey(EmployeeKey)**.



JNDI Explorer

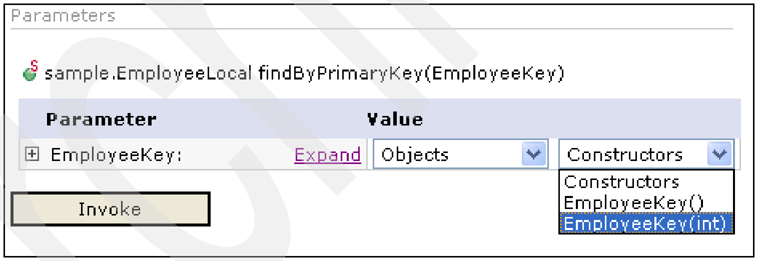
⊟ EJB Beans

- ⊟ EmployeeLocal
  - ⊟ EmployeeLocalHome
    - Method Visibility
    - ● EmployeeLocal create(int)
    - ● EmployeeLocal create(int, String, Strin
    - ● EmployeeLocal findByPrimaryKey(Emp

i No UserTransaction loaded

4. From the resulting right pane, perform the following:

   – Change the right most selection from **Constructors** to **EmployeeKey(int)**.



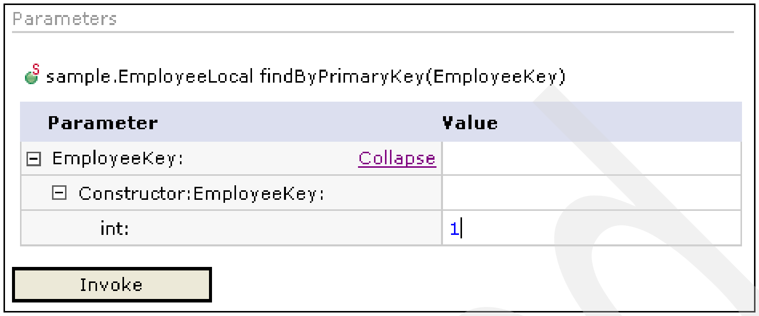
Parameters

sample.EmployeeLocal findByPrimaryKey(EmployeeKey)

Parameter		Value	
⊞ EmployeeKey:	Expand	Objects	Constructors

Constructors
EmployeeKey()
EmployeeKey(int)

– Expand the left most selection **to EmployeeKey** → **Constructor:EmployeeKey** → **int**. Enter “1” under the **Value** column. Click **Invoke**.

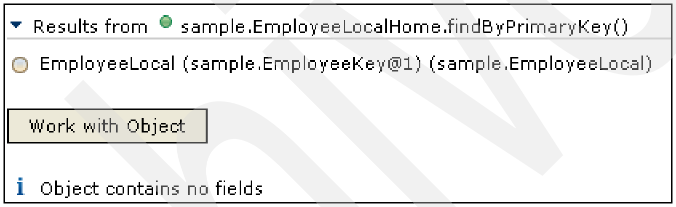


Parameters

sample.EmployeeLocal findByPrimaryKey(EmployeeKey)

Parameter	Value
⊟ EmployeeKey: Collapse	
⊟ Constructor:EmployeeKey:	
int:	1

The operation will complete with a message similar to the following 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 shown in the following image:



▾ Results from ● sample.EmployeeLocalHome.findByPrimaryKey()

○ EmployeeLocal (sample.EmployeeKey@1) (sample.EmployeeLocal)

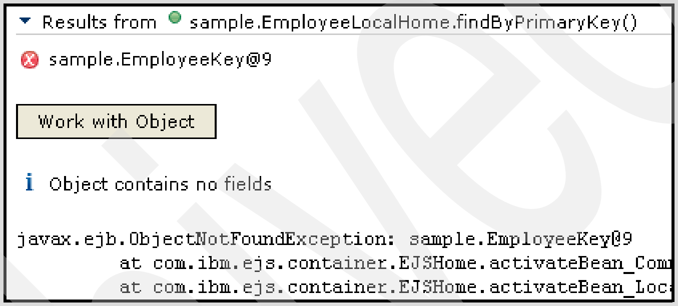
ℹ Object contains no fields

– Work with the returned object (for primary key = 1) by clicking **Work with object**.

Behind the scenes:

The entity EJB uses a complex type (EmployeeKey, found **at EJB Projects → DonateEJB → ejbModule → sample → EmployeeKey.java**) as the primary key for the Employee record, and the int employeeid is an attribute of type. This appears to be somewhat cumbersome for entity EJBs with a single key field, but utilizing a complex type simplifies the case where entity EJBs have multiple key fields.

VARIATION: If the instance could not be located (for example, if you had tried locating the object with a primary key that does not exist, such as 9), the result would be similar to the following image.



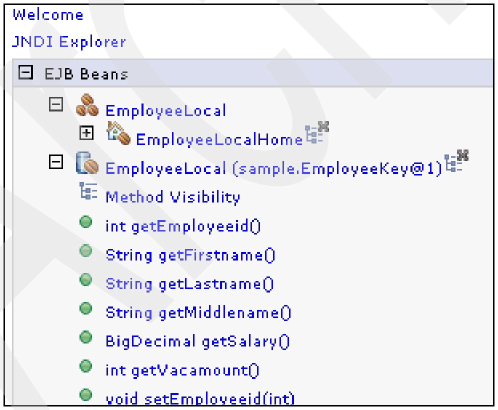
Results from ● sample.EmployeeLocalHome.findByPrimaryKey()

☒ sample.EmployeeKey@9

i Object contains no fields

```
javax.ejb.ObjectNotFoundException: sample.EmployeeKey@9
        at com.ibm.ejs.container.EJSHome.activateBean_Com
        at com.ibm.ejs.container.EJSHome.activateBean_Loc
```

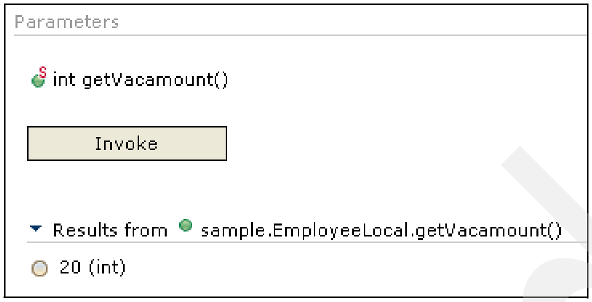
5. Work with the entity EJB instance by expanding on the left to **EJB Beans → EmployeeLocal(sample.EmployeeKey@1)**.



– Select a getter method (such as **int getVacamount**) from the left pane.

- From the right pane, click **Invoke**.

- View the result (in the bottom half of the right hand pane).



Parameters

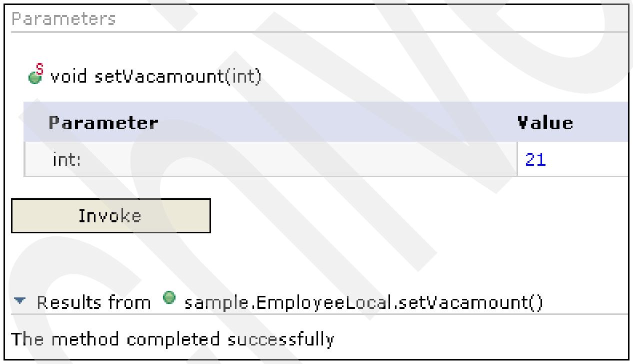
int getVacamount()

Results from sample.EmployeeLocal.getVacamount()

20 (int)

– Select a setter method (such as **void setVacamount**) from the left hand pane.

- From the right hand pane, under **Value**, enter a new value (such as 21).
- Click **Invoke**.



Parameters

void setVacamount(int)

Parameter	Value
int:	21

Results from sample.EmployeeLocal.setVacamount()

The method completed successfully

– Optional step: Re-execute the getter method (as described above), and verify that the value changed.

– Close the UTC.

Extra credit:

Updates to entity EJBs are automatically written to the back-end persistence database when the associated transaction is committed. You do not have to write any unique code to do this.

In the above test, each action within the UTC is run as a separate transaction. Therefore, updates are made each time you selected invoke.

If you want to verify that the update was successfully made to the database, you must first stop **ExperienceJ2EE Server @localhost** because Derby only allows a single Java process to access the database. Following are the basic steps you would follow:

1. From the **Servers** view in the lower-right pane, select **ExperienceJ2EE Server @localhost**, and right-click **Stop**.
2. From the **Database Explorer** view in the lower- right pane, perform the following:
   a. Select **Connections** → **Vacation**, and right-click **Reconnect**.
   b. At the **Database Authorization** pop-up, click **OK** (no password is required).
   c. Select **Connections** → **Vacation[Derby 10.1]** → **Vacation** → **Schemas** → **EXPERIENCEJ2EE** → **Tables** → **EMPLOYEE**, and right-click **Data** → **Sample Contents**.
3. At the resulting **Data Output** view in the lower-right pane, note the updated vacation amount for employee ID 1 (assuming that you made the changes outlined above).
4. Switch back to the **Database Explorer** view, select **Connections** → **Vacation[Derby 10.1]**, and right-click **Disconnect**.
5. From the **Servers** view in the lower right pane, select **ExperienceJ2EE Server @localhost**, and right-click **Start**.
6. Switch to the **Console** view in the lower-right pane, and ensure that the server starts without errors.

If the **Database Explorer** view does not appear, or if the Vacation database is not in the list of connections, repeat the steps in section 4.3.1, “Add the database to the Database Explorer view” on page 75.

5.7 Explore!

- **Finder and select methods**. The default method for locating an instance of an entity EJB is findByPrimaryKey, which returns a single object that represents that specific primary key. However, what if you need to locate an object based on a more complex query search, something more complex than finding a single record based on the primary key?

  The answer is that the EJB finder and select methods allow you to locate an object, multiple objects, or specific values from an object based on a query defined in the EJB deployment descriptor.

  The queries are implemented using the EJB query language (EJB QL). The EJB QL syntax is defined in the EJB 2.1 specification.

  – A finder method is externally available through the entity EJB home interface and can return one or more EJB instances.

    For example, the following definition (created in ejb-jar.xml either manually or through the EJB deployment descriptor editor) returns all instances of the entity EJB where the vacation value is greater than the input:

    ```
    <query>
       <description></description>
       <query-method>
          <method-name>findVacationGreaterThan</method-name>
          <method-params>
             <method-param>java.lang.Integer</method-param>
          </method-params>
       </query-method>
       <ejb-ql>select object(o) from Employee o where o.vacamount
    &gt; ?1</ejb-ql>
    </query>
    ```

    A finder method must start with find (such as findVacationGreaterThan).

  – A select method is private (only available to other methods in the entity EJB class) and can return one or more EJB instances or values from an EJB. This private method can be wrapped by another custom method and made available to an external requester.

    For example, the following definition returns a collection of all first name values from the back end database:

    ```
    <query>
          <description></description>
       <query-method>
          <method-name>ejbSelectAll</method-name>
          <method-params>
          </method-params>
    ```

```
        </query-method>
        <ejb-ql>select o.firstname from Employee o</ejb-ql>
    </query>
```

A select method must start with ejbSelect (such as ejbSelectAll).

- **Bean managed persistence**. The Employe entity EJB accessed records in a back-end relational database. This is called container managed persistence (CMP) and is the most common type of entity EJB implementation. However, what if you need to access data in a record oriented format that is not stored in a relational database?

  The answer is that an entity EJB implemented using bean managed persistence (BMP) can access information stored in a non-relational record oriented format. The developer defines the BMP entity EJB, and then codes the various methods required to read and write the information from the back-end storage format (such as an XML file, for example).

  Contrast this with CMP where the developer defines the entity EJB (using the AST V6.1 EJB wizard) and then generates the runtime code using the EJB deploy operation.

- **Container managed relationships**. User data is often maintained in multiple tables in a relational database, and specific relationships are defined to ensure that these separate databases are consistent. For example, a user table may contain a department number field, and this same field is used as the primary key in the department table. How do you ensure that EJBs maintain this type of relationship?

  The answer is that container managed relationships allow you to define relationships between various entity EJBs. Specific relationships include one-to-one, one-to-many, and many-to-many.

  The developer defines these relationships in the EJB deployment descriptor, and the EJB runtime container enforces them.

- **Access Intents**. One of the most important best practices for using relational databases is to tune your access—pick the appropriate locking, caching, and read-ahead strategies. For example, if your access to the database is primarily read-only, you may choose to implement an optimistic locking approach—meaning that the database record is not locked. How do you optimize database access for a CMP entity EJB?

  The answer is that IBM implemented an EJB extension called access intents that allows you to specify these policies in the EJB deployment descriptor.

Additional explore resources are available in the EJB 2.1 - Final Release specification listed in the learn resources, and at the following Web sites:

- The J2EE 1.4 Tutorial: Chapter 26: Bean-Managed Persistence Examples:

  `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

- IBM developerWorks: Understanding WebSphere Application Server EJB access intents:

  `http://www.ibm.com/developerworks/websphere/techjournal/0406_persson/0406_persson.html`

- WAS V6.1 InfoCenter: Naming and directory:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/welc6tech_nam.html`

6

Create the funds data access element

In this chapter you create a new entity EJB, and then use this definition to create a new database in a *top-down* scenario.

6.1 Learn!

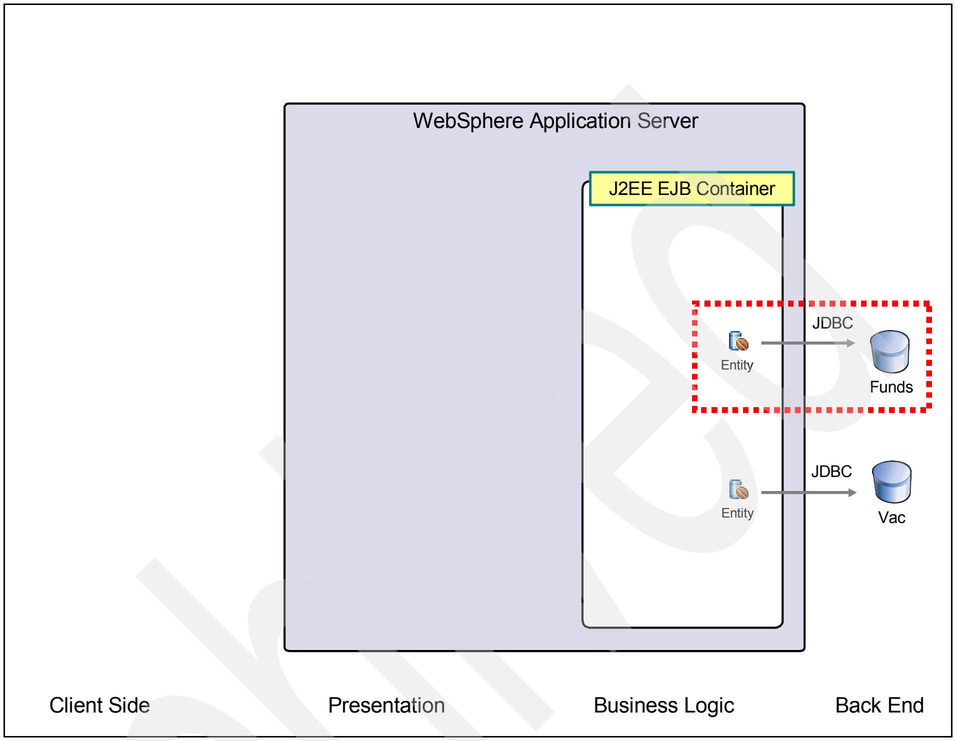


Figure 6-1 Donate application: Funds entity EJB

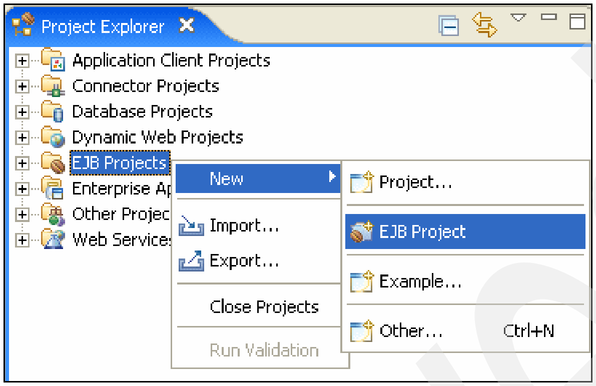
This chapter implements the same basic entity EJB used in Chapter 5, “Create the employee data access element” on page 83. The key difference is that here you are employing a *top-down* implementation (starting with entity EJB), whereas before you performed a *bottom-up* implementation (starting with the database).

Refer to section 5.1, “Learn!” on page 84 for introductory information on entity EJBs.

6.2 Create the new EJB project

You must create the Funds entity EJB in a different EJB project from the Employe entity EJB because they use separate databases. The J2EE specification maps all the entity EJBs in a single EJB module to the same database; therefore, you cannot have the Employee entity EJB pointing to one database and the Funds entity EJB pointing to another database in the same module (project).

1. From the **Project Explorer**, select **EJB Projects**, and right-click **New** → **EJB Project**.



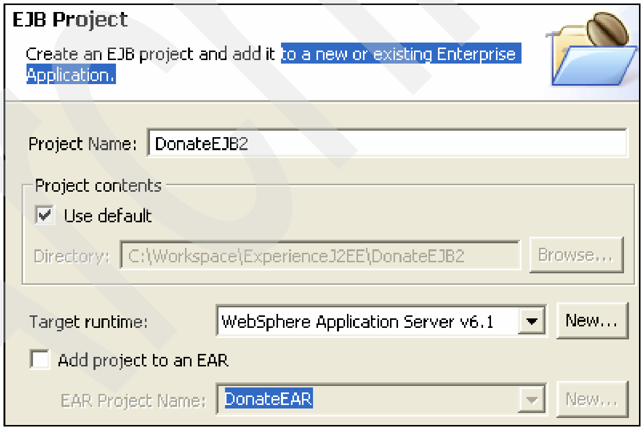
2. At the **EJB Project** pop-up, perform the following:

   – **Name**: **DonateEJB2**

   – **Target Runtime**: **WebSphere Application Server v6.1**

   – Clear **Add project to an EAR**.

   – Click **Next**.



EJB Project

Create an EJB project and add it to a new or existing Enterprise Application.

Project Name: DonateEJB2

Project contents

☑ Use default

Directory: C:\Workspace\ExperienceJ2EE\DonateEJB2 Browse...

Target runtime: WebSphere Application Server v6.1 New...

☐ Add project to an EAR

EAR Project Name: DonateEAR New...

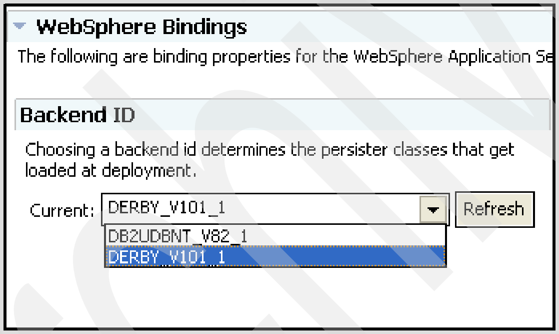
Behind the scenes:

You will add DonateEJB2 to the DonateEAR Enterprise Application in section 6.6, “Add DonateEJB2 to DonateEAR” on page 156. However, you are deferring this step until after you generate the proper mapping and deployment code.

If you added DonateEJ2 to DonateEAR now, before generating the proper mapping and deployment code, the automatic publishing to the test server would attempt to generate these defaults. One of the defaults is to create a back-end mapping for DB2, and you will use Derby.

This would not cause a failure because an application can have multiple back-end mappings, but it would populate the project with unneeded classes and folders.

If you have an EJB project with multiple back-end mappings, you set the active one in the deployment descriptor on the Overview tab.


WebSphere Bindings

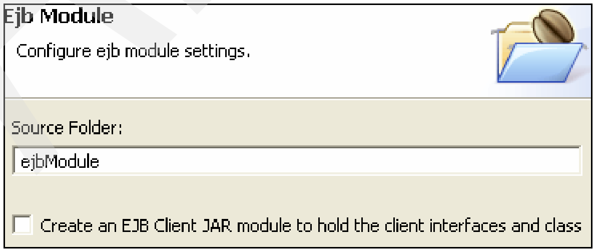
The following are binding properties for the WebSphere Application Se

Backend ID

Choosing a backend id determines the persister classes that get loaded at deployment.

Current: DERBY_V101_1

3. At the **Select Project Facets** pop-up, click **Next**.
4. At the **EJB Module** pop-up, clear **Create an EJB Client jar module...**
5. Click **Finish**.


Ejb Module

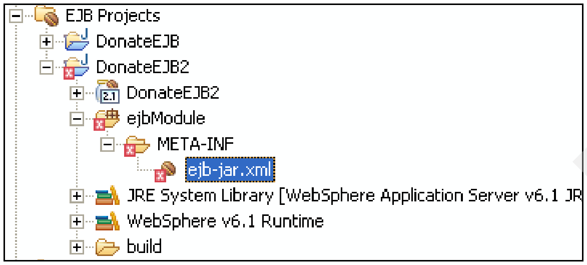
Configure ejb module settings.

Source Folder:

ejbModule

□ Create an EJB Client JAR module to hold the client interfaces and class

Note the new **DonateEJB2** entry in the **Project Explorer** view.



Behind the scenes:

Note the red error symbols in the **DonateEJB2** project in the **Project Explorer** view, and the associated error message listed in the **Problems** view (in the lower right pane):

```
cvc-complex-type.2.4.b: The content of element 'ejb-jar' is not
complete. One of '{"http://java.sun.com/xml/ns/j2ee":display-name,
"http://java.sun.com/xml/ns/j2ee":icon,
"http://java.sun.com/xml/ns/j2ee":enterprise-beans}' is expected.
```

This error occurs because the **DonateEJB2** module is currently invalid because it does not contain any EJB artifacts. This error is resolved after you create the Funds entity EJB.

6.3 Develop the Funds entity EJB

In this section you create the Funds entity EJB using the Create an Enterprise Bean wizard.

6.3.1 Define the Funds entity EJB

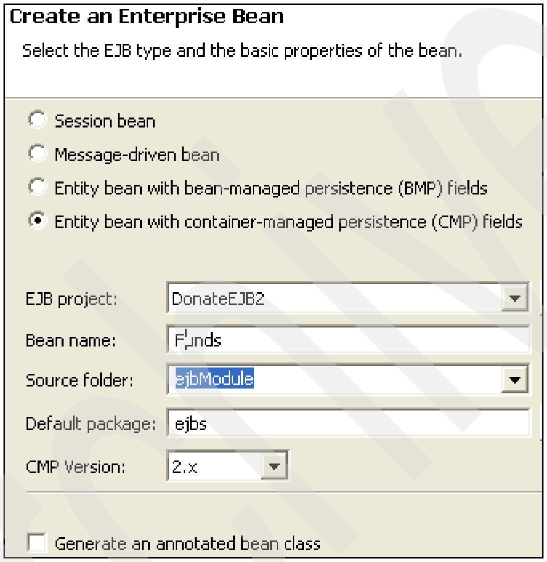
Since this is a top-down scenario, you must define the entity EJB as the starting point. This is in contrast with the bottom-up scenario used in the preceding chapter where the wizard automatically created the entity EJB.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Entity Beans**.

Note: Make sure you select **DonateEJB2** and NOT **DonateEJB**!

2. Right-click **New** → **Entity Bean**.

3. At the **Create an Enterprise Bean** pop-up perform the following tasks:
   - Select **Entity bean with container managed persistence (CMP) fields**
   - **EJB project**: **DonateEJB2**
   - **Bean name**: **Funds**
   - **Source Folder**: **ejbModule**
   - **Default package**: **ejbs**
   - **CMP Version**: **2.x**
   - Clear the **Generate an annotated bean class** field.
   - Click **Next**.



Create an Enterprise Bean

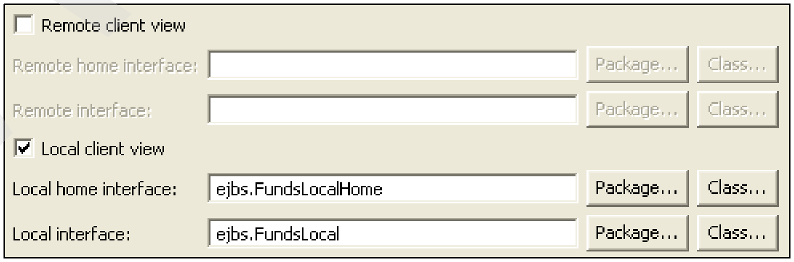
Select the EJB type and the basic properties of the bean.

○ Session bean
○ Message-driven bean
○ Entity bean with bean-managed persistence (BMP) fields
● Entity bean with container-managed persistence (CMP) fields

EJB project: DonateEJB2
Bean name: Funds
Source folder: ejbModule
Default package: ejbs
CMP Version: 2.x

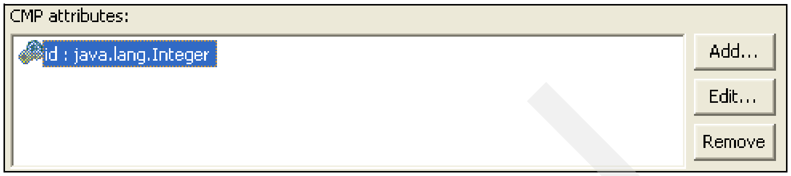
□ Generate an annotated bean class

4. At the **Enterprise Bean Details** pop-up, perform the following actions:
   - Ensure that the **Remote client view** is cleared and that the **Local client view** is selected. These correspond to the Remote and Local EJB interfaces, respectively.



□ Remote client view
Remote home interface:
Remote interface:
☑ Local client view
Local home interface: ejbs.FundsLocalHome
Local interface: ejbs.FundsLocal

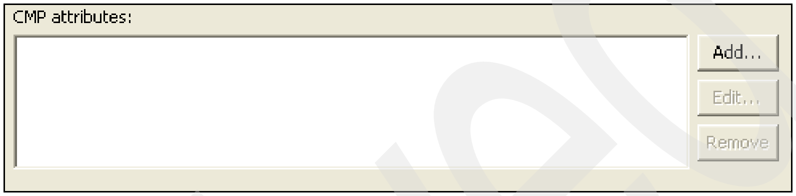
- In the **CMP attributes** panel, near the bottom of the **Enterprise Bean Details** pop-up, select the existing attribute **id: java.lang.Integer**, and click **Remove**.



CMP attributes:

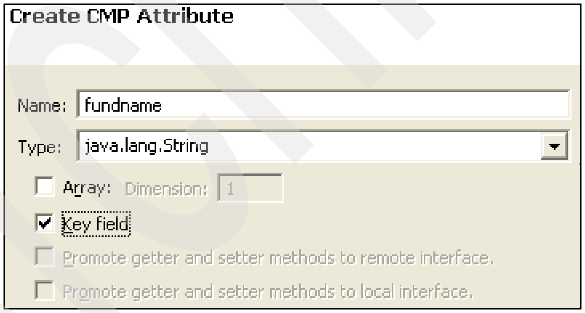
id : java.lang.Integer

- Next to the now empty **CMP attributes** panel, click **Add...**.



CMP attributes:

- At the **Create CMP Attribute** pop-up, populate the following fields:
  - **Name**: **fundname**
  - **Type**: **java.lang.String** (select from pull-down)
  - Select **Key field**.
  - Click **Apply**. The **Create CMP Attribute** pop-up appears again.



Create CMP Attribute

Name: fundname

Type: java.lang.String

☐ Array: Dimension: 1

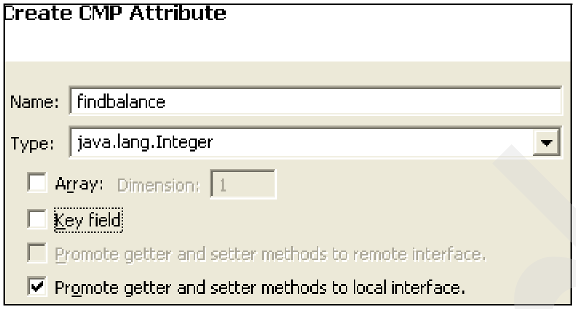
☑ Key field

☐ Promote getter and setter methods to remote interface.

☐ Promote getter and setter methods to local interface.

5. At the **Create CMP Attribute** pop-up, perform the following tasks:
   - **Name**: **fundbalance**
   - **Type**: **java.lang.Integer** (select from pull-down)
   - Clear the **Key field**.
   - Select **Promote getter and setter methods to local interface**.

- Click **Apply**, and then click **Close**.



Create CMP Attribute

Name: findbalance

Type: java.lang.Integer

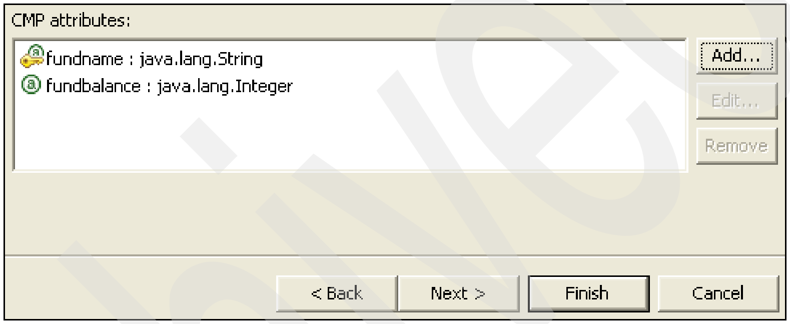
□ Array: Dimension: 1

□ Key field

□ Promote getter and setter methods to remote interface.

☑ Promote getter and setter methods to local interface.

6. Return to the **Enterprise Bean Details** pop-up, and click **Finish**.



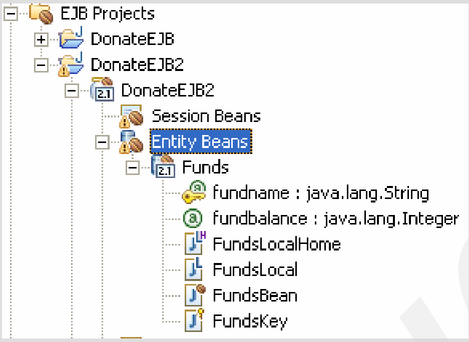
CMP attributes:

fundname : java.lang.String

fundbalance : java.lang.Integer

Behind the scenes:

You can view the definition of the newly created Funds entity EJB from the **Project Explorer** view by expanding to **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Entity Beans** → **Funds**:

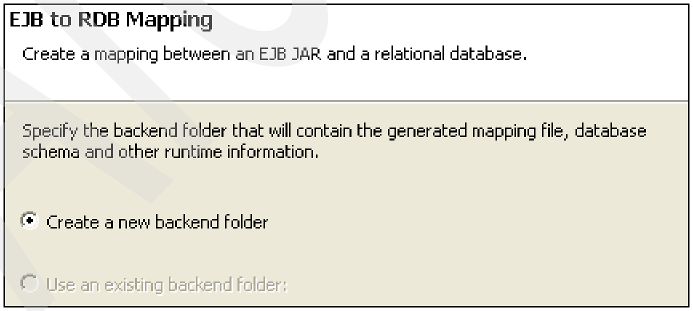


```
EJB Projects
  DonateEJB
  DonateEJB2
    DonateEJB2
      Session Beans
      Entity Beans
        Funds
          fundname : java.lang.String
          fundbalance : java.lang.Integer
          FundsLocalHome
          FundsLocal
          FundsBean
          FundsKey
```

You can also view this information by opening the DonateEJB2 deployment descriptor by selecting **EJB Projects** → **DonateEJB2** → **DonateEJB2** and double-clicking to open, or by opening **EJB Projects** → **DonateEJB2** → **ejbModule** → **META-INF** → **ejb-jar.xml**.

6.3.2 Map the Funds entity EJB to a database

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2**, and right-click **EJB to RDB Mapping** → **Generate Map**.
2. At the **EJB to RDB Mapping** pop-up, select **Create a new backend folder**, and click **Next**.



EJB to RDB Mapping

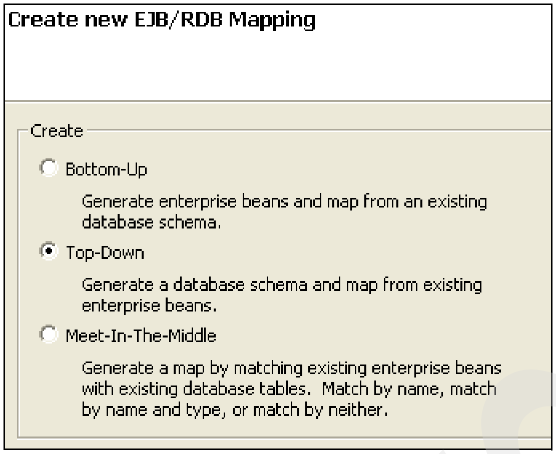
Create a mapping between an EJB JAR and a relational database.

Specify the backend folder that will contain the generated mapping file, database schema and other runtime information.

● Create a new backend folder

○ Use an existing backend folder:

3. At the **Create EJB/RDB Mapping** pop-up, select **Top-Down**, and click **Next**.



Create new EJB/RDB Mapping

Create

○ Bottom-Up

Generate enterprise beans and map from an existing database schema.

● Top-Down

Generate a database schema and map from existing enterprise beans.

○ Meet-In-The-Middle

Generate a map by matching existing enterprise beans with existing database tables. Match by name, match by name and type, or match by neither.

4. At the second **Create EJB/RDB Mapping** pop-up, perform the following actions:

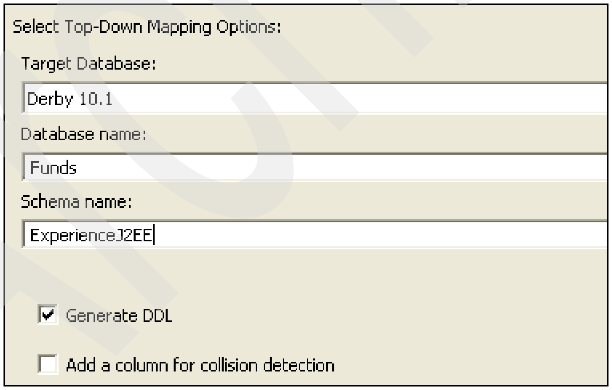
   – **Target Database**: **Derby 10.1**

   – **Database name**: **Funds**

   – **Schema name**: **ExperienceJ2EE**

   – Select **Generate DDL**.

   – Click **Finish**.



Select Top-Down Mapping Options:

Target Database: Derby 10.1

Database name: Funds

Schema name: ExperienceJ2EE

☑ Generate DDL

□ Add a column for collision detection

Behind the scenes:

You are setting the Schema name to something other than NULLID. Otherwise, since you set a JAAS authentication alias on the data source, WAS V6.1 would attempt to use the user name from the JAAS authentication alias, and access to the Funds database would fail with a message similar to:

```
CNTR0019E: EJB threw an unexpected (non-declared) exception
during invocation of method "findByPrimaryKey". Exception data:
com.ibm.ws.exception.WsEJBException
...
...
Caused by: java.sql.SQLException: Schema 'J2EEADMIN' does not
existDSRA0010E: SQL State = 42Y07, Error Code = 20,000
...
...
```

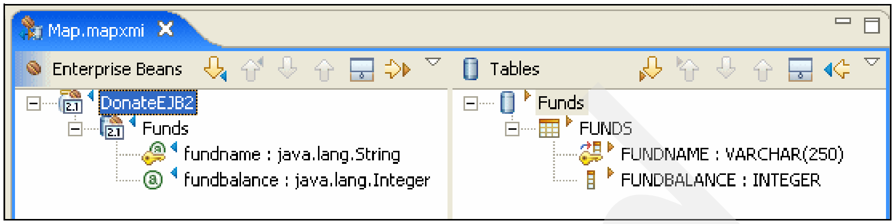
The following occurs when you set the schema to something other than NULLID:

- The AST V6.1tooling generates a schema file (table.ddl) that fully qualifies the table name (ExperienceJ2EE.FUNDS).
- The Derby ij tool creates a table under this qualifier when you run it against table.ddl.
- The WAS V6.1 runtime uses this qualifier when accessing the tables for the entity EJB.

However, if you do not set the schema to something other than NULLID, the following occurs:

- The AST V6.1 tooling generates a schema file with a non-qualified table name (FUNDS).
- The Derby ij tool creates a table under the default qualifier of APP when you run it against table.ddl.
- The WAS V6.1 runtime, since the qualified was not specified, accesses the database with the user ID from the JAAS authentication alias, and access fails because the user ID is J2EEADMIN in this case, and the qualifier/schema in the database is APP.

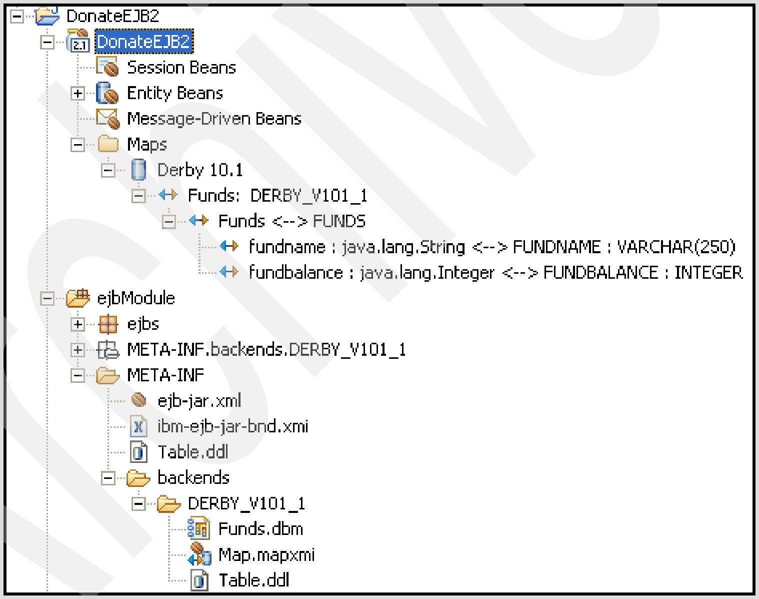
5. The mapping file (**Map.mapxmi**) automatically opens. View the mapping between the CMP fields of the EJB and the table fields. Close the file when finished.



```
Map.mapxmi
Enterprise Beans                          Tables
DonateEJB2                                Funds
  Funds                                     FUNDS
    fundname : java.lang.String               FUNDNAME : VARCHAR(250)
    fundbalance : java.lang.Integer           FUNDBALANCE : INTEGER
```

Behind the scenes:

The backend-mapping is visible in the **Project Explorer** view through the abstract deployment descriptor view under **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Maps** or through the artifact view under **EJB Projects** → **ejbModule** → **META-INF** → **backends**.



```
DonateEJB2
  DonateEJB2
    Session Beans
    Entity Beans
    Message-Driven Beans
    Maps
      Derby 10.1
        Funds:  DERBY_V101_1
          Funds <--> FUNDS
            fundname : java.lang.String <--> FUNDNAME : VARCHAR(250)
            fundbalance : java.lang.Integer <--> FUNDBALANCE : INTEGER
  ejbModule
    ejbs
    META-INF.backends.DERBY_V101_1
    META-INF
      ejb-jar.xml
      ibm-ejb-jar-bnd.xmi
      Table.ddl
      backends
        DERBY_V101_1
          Funds.dbm
          Map.mapxmi
          Table.ddl
```

The Table.ddl file contains the schema (table definitions) needed for the EJB. We use this file in the next section to create the tables in the database.

```
CREATE TABLE ExperienceJ2EE.FUNDS (
      FUNDNAME VARCHAR(250) NOT NULL,
      FUNDBALANCE INTEGER
   )
;

ALTER TABLE FUNDS ADD CONSTRAINT PK_FUNDS PRIMARY KEY (FUNDNAME)
;
```

If needed, the schema file can be (re)generated by selecting the EJB project and right-clicking **EJB to RDB Mapping** → **Generate Schema DDL**.

6.3.3 Set the Default CMP connection factory JNDI name

The following steps are similar to those used to set the Default CMP connection factory JNDI name for DonateEJB in section 5.3.3, “Set the Default CMP connection factory JNDI name” on page 98.

1. From the J2EE Perspective **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** and double-click to open the EJB deployment descriptor.
2. From the resulting **EJB Deployment Descriptor** editor located in the upper central window, ensure that you are on the **Overview** tab.
   - Scroll down to **WebSphere Bindings**, and on the right under **JNDI - CMP Connection Factory Binding** change the **JNDI name** from **jdbc/Default** to **jdbc/Funds**.
   - Save and close the **EJB Deployment Descriptor**.

6.3.4 Generate the deployment code

The following steps are similar to the those used to generate the deployment code for DonateEJB in section 5.3.4, “Generate the deployment code” on page 98.

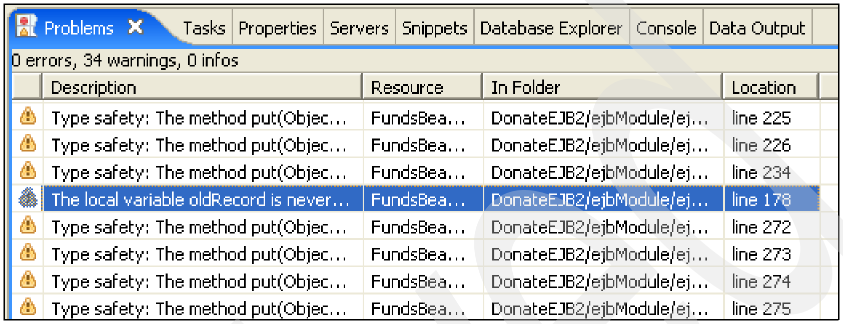
1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2**, and right-click **Deploy**.

   The resulting pop-up automatically closes when the operation is completed.

6.3.5 Alter the workspace settings to ignore specific warnings

Warnings appear in the **Problems** view (in the lower-right pane) after generating the EJB deploy code.

- Most warnings indicate a type safety check.
- Some warnings indicate that local variables are defined but never used.



0 errors, 34 warnings, 0 infos

Description	Resource	In Folder	Location
Type safety: The method put(Objec...	FundsBea...	DonateEJB2/ejbModule/ej...	line 225
Type safety: The method put(Objec...	FundsBea...	DonateEJB2/ejbModule/ej...	line 226
Type safety: The method put(Objec...	FundsBea...	DonateEJB2/ejbModule/ej...	line 234
The local variable oldRecord is never...	FundsBea...	DonateEJB2/ejbModule/ej...	line 178
Type safety: The method put(Objec...	FundsBea...	DonateEJB2/ejbModule/ej...	line 272
Type safety: The method put(Objec...	FundsBea...	DonateEJB2/ejbModule/ej...	line 273
Type safety: The method put(Objec...	FundsBea...	DonateEJB2/ejbModule/ej...	line 274
Type safety: The method put(Objec...	FundsBea...	DonateEJB2/ejbModule/ej...	line 275

For simplicity of development, we ignore these errors. In an ideal Java 5 implementation you could address them by using coding practices that fully exploit the new Java 5 features, but you will choose to accept these (and run the generated code as-is).

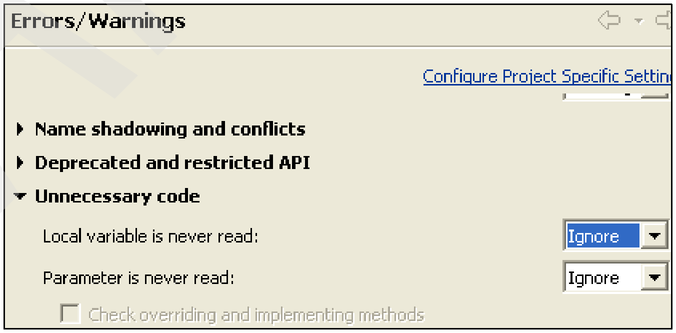
1. From the AST V6.1 **J2EE Perspective** action bar, select **Window** → **Preferences**.

   The **Preferences** pop-up appears.

2. At the Preferences pop-up, perform the following:

   – On the left of the **Preferences** pop-up, select **Java** → **Compiler** → **Errors/Warnings**.

   – On the right (at the resulting **Errors/Warnings** pane), expand the **Unnecessary code** section, and change the option for **Local variable is never read** from Warning to **Ignore**.



Errors/Warnings

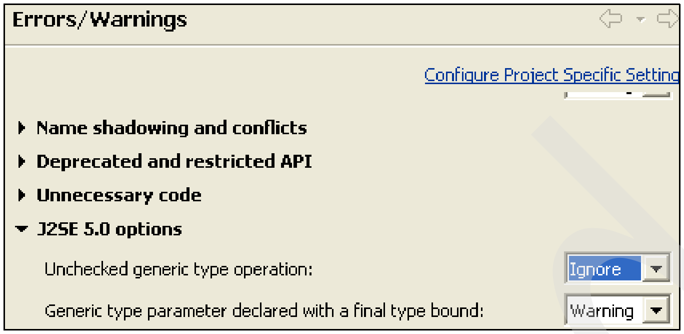
Configure Project Specific Settin
Name shadowing and conflicts
Deprecated and restricted API
Unnecessary code

Local variable is never read: Ignore

Parameter is never read: Ignore

□ Check overriding and implementing methods

- Expand the **J2SE™ 5.0** section, and change the option for **Unchecked generic type operation** from **Warning** to **Ignore**.



Errors/Warnings

Configure Project Specific Setting

- Name shadowing and conflicts
- Deprecated and restricted API
- Unnecessary code
- J2SE 5.0 options

Unchecked generic type operation: Ignore

Generic type parameter declared with a final type bound: Warning

- Click **OK**.
- At the resulting **Error/Warnings Settings Changed** pop-up, click **Yes**.

The **Problems** view should now show no errors or warnings.

6.4 Create the Funds database and tables

In this section you create the required database and tables:

- You create the Funds database using the New Connection wizard in AST V6.1. This option is available for simple “load and go” databases like Derby and Cloudscape. AST V6.1 cannot create databases for other database managers, such as DB2, Informix®, Oracle Database, Microsoft SQL Server, and Sybase.

  Derby and Cloudscape also support creating the databases outside of AST V6.1. Recall that you used this support when you created the Vacation Derby database from a command line in section 4.2, “Create the Vacation application database” on page 72.

- You create the required tables by using the derby ij command line interface to execute the table.ddl definition created in section “6.3.2 Map the Funds entity EJB to a database” on page 143.

6.4.1 Create the Funds database

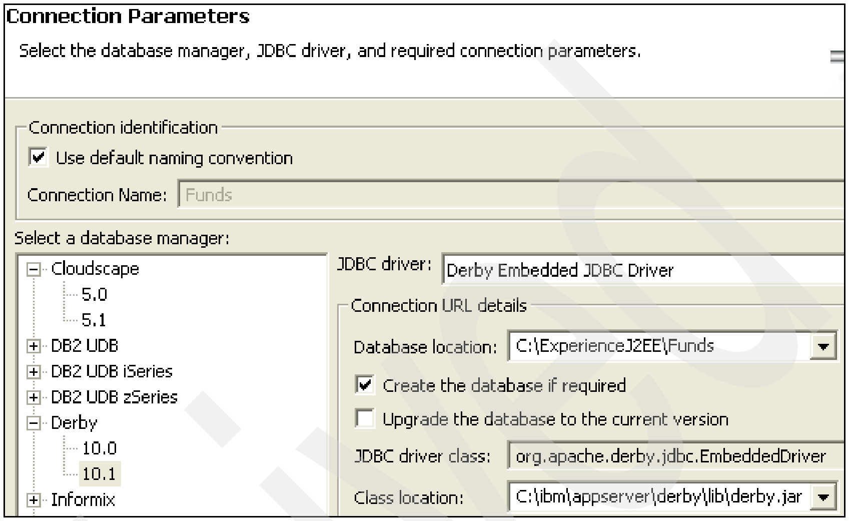
1. Switch to the **Database Explorer** view located in the lower-right pane. Select **Connections**, and right-click **New Connection...**.

> **Off course?**
>
> If the **Database Explorer** view is not open (not visible as one of the selections in the lower-right pane), you can re-open it using the following steps.
>
> - From the AST V6.1 **J2EE Perspective** action bar, select **Window** → **Show View** → **Other**.
> - From the **Show View** pop-up, select **Data** → **Database Explorer**, and click **OK**.
>
> You opened this view in section 4.3.1, “Add the database to the Database Explorer view” on page 75, but perhaps you closed it or reset the perspective (from the action bar **Window** → **Reset Perspective**).
>
> You can also save the current combination of panes and views as a new Perspective, or overwrite an existing one, from the AST V6.1 application action bar **Window -> Save Perspective As** pull-down.
>
> In this case, you could save the current configuration (with the open **Data Explorer** view) as J2EE, overwriting the default J2EE Perspective configuration, or under a new name.

2. At the **Connection Parameters** pop-up perform the following:
   - **Select a database manager**: Expand to/select **Derby** → **10.1**
   - **JDBC Driver**: **Derby Embedded JDBC Driver**
   - Windows **Database location**: **C:\ExperienceJ2EE\Funds**
   - Linux **Database location**: **/home/*<user>*/ExperienceJ2EE/Funds**

     where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.
   - Windows **Class location**: **C:\IBM\AppServer\derby\lib\derby.jar**
   - Linux **Class location**: **/opt/IBM/AppServer/derby/lib/derby.jar**
   - Select **Create the database if required**.

- Click **Test Connection**. If you entered the information properly, you will receive a pop-up stating **Connection to Derby is successful**. Click **OK** to close this **Test Connection** pop-up.
- At the **Connection Parameters** pop-up, click **Finish**.



Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: Funds

Select a database manager:

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\ExperienceJ2EE\Funds

☑ Create the database if required

□ Upgrade the database to the current version

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: C:\ibm\appserver\derby\lib\derby.jar

3. Disconnect from the Funds database from the **Database Explorer** view by selecting **Connections** → **Funds**, and right-clicking **Disconnect**.

6.4.2 Create the Derby Tables

1. Open a command line (appropriate for your operating system) and run the following commands:

   - Windows

     ```
     C:
     CD C:\Workspace\ExperienceJ2EE\DonateEJB2\ejbModule\META-INF
     \IBM\AppServer\derby\bin\embedded\ij
     ```

     and at the ij command prompt execute:

     ```
     connect 'jdbc:derby:c:\ExperienceJ2EE\Funds';
     run 'Table.ddl';
     exit;
     ```

   - Linux

     ```
     cd
     $HOME/workspace/ExperienceJ2EE/DonateEJB2/ejbModule/META-INF
     /opt/IBM/AppServer/derby/bin/embedded/ij.sh
     ```

and at the ij command prompt execute:

```
connect 'jdbc:derby:/home/<user>/ExperienceJ2EE/Funds';
run 'Table.ddl';
exit;
```

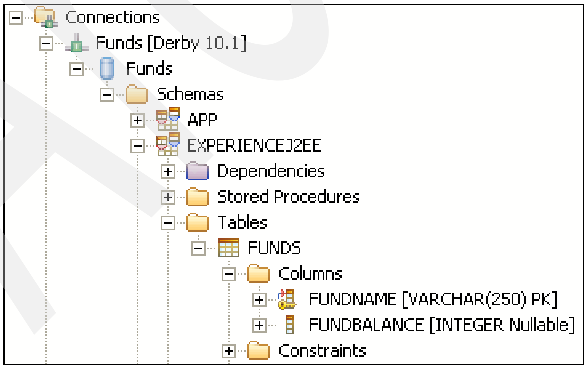
where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.

The following messages should be generated when running table.ddl:

```
ij> CREATE TABLE FUNDS (
                FUNDNAME VARCHAR(250) NOT NULL,
                FUNDBALANCE INTEGER
        )
;
0 rows inserted/updated/deleted
ij> ALTER TABLE FUNDS ADD CONSTRAINT PK_FUNDS PRIMARY KEY
(FUNDNAME)
;
0 rows inserted/updated/deleted
```

6.4.3 Verify that the tables were created

1. In AST V6.1, in the **Database Explorer** view (in the lower-right pane), select **Connections** → **Funds**, and **right-click** → **Reconnect**.
2. At the **Database Authorization** pop-up, click **OK**.
3. Back in the **Database Explorer** view, fully expand the **Connections** → **Funds** subtree to view the created tables. The tables for the Funds entity EJB were created under the **ExperienceJ2EE** schema.



```
Connections
  Funds [Derby 10.1]
    Funds
      Schemas
        APP
        EXPERIENCEJ2EE
          Dependencies
          Stored Procedures
          Tables
            FUNDS
              Columns
                FUNDNAME [VARCHAR(250) PK]
                FUNDBALANCE [INTEGER Nullable]
              Constraints
```

4. From the **Database Explorer** view, select **Connections** → **Funds**, and right-click **disconnect**.

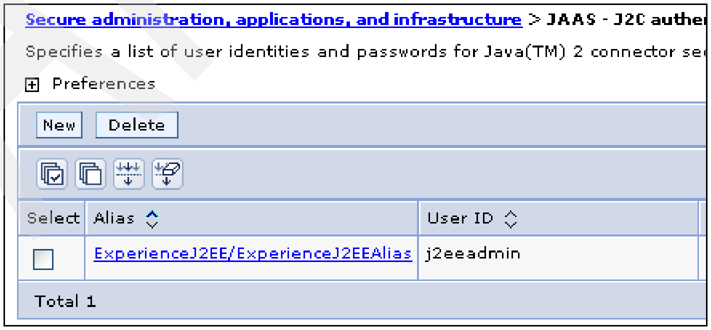
6.5 Configure the test server for DonateEJB2

The ExperienceJ2EE Server @ localhost test server serves as your J2EE runtime environment. You must configure it with the proper JDBC provider and JDBC data source to map from the DonateEJB2 project to the Funds database/table.

6.5.1 Verify that the ExperienceJ2EE JAAS authentication alias exists

The Funds database uses the same **ExperienceJ2EE/ExperienceJ2EEAlias** JAAS authentication alias created in section 5.4.2, “Create the ExperienceJ2EE JAAS authentication alias” on page 105.

1. Start the **Admin Console** using the same steps as described in section “5.4.1 Start the admin console” on page 102.
   – Recall that at the **Admin Console** login screen you will use the **j2eeadmin** user ID and password that you defined in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51.
2. On the left side of the **Admin Console**, expand to **Security**, and click on **Secure administration, applications, and infrastructure**.
3. On the right (in **Secure administration, applications, and infrastructure**), under **Authentication**, expand **Java authentication and Authorization Service**. Select **J2C authentication data**.
4. On the right **(in Secure administration, applications, and infrastructure → JAAS → J2C authentication data)**, verify that the **ExperienceJ2EE/ExperienceJ2EEAlias** authentication alias exists:
   – If it does exist, continue to section 6.5.2, “Verify that the Derby (XA) provider already exists” on page 154.
   – If it does NOT exist, create it by following the steps in section “5.4.2 Create the ExperienceJ2EE JAAS authentication alias” on page 105.



Secure administration, applications, and infrastructure > JAAS - J2C authe

Specifies a list of user identities and passwords for Java(TM) 2 connector se

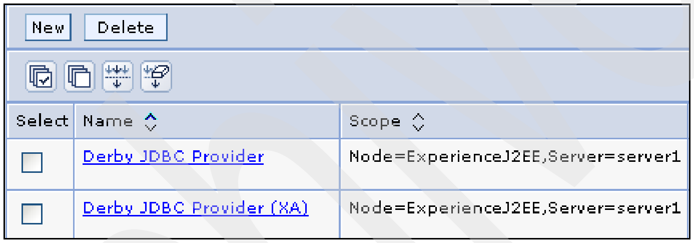
Select	Alias	User ID
□	ExperienceJ2EE/ExperienceJ2EEAlias	j2eeadmin

Total 1

6.5.2 Verify that the Derby (XA) provider already exists

The Funds database uses the same **Derby JDBC Provider (XA) JDBC Provider** created in section 5.4.3, “Create the Derby XA JDBC provider” on page 107.

1. From the left side of the **Admin Console**, select **Resources**, and click **JDBC providers**.
2. On the right at the **JDBC providers** panel, under **Scope** use the pull-down to change the scope from **All scopes** to **Node=ExperienceJ2EE, Server=server1**.
3. Verify that the **Derby JDBC Provider (XA) provider** exists:
   - If it does exist, continue to section 6.5.3, “Create the Funds JDBC Data source” on page 154.
   - If it does NOT exist, create it by following the steps in section “5.4.3 Create the Derby XA JDBC provider” on page 107.



Select	Name	Scope
□	Derby JDBC Provider	Node=ExperienceJ2EE,Server=server1
□	Derby JDBC Provider (XA)	Node=ExperienceJ2EE,Server=server1

6.5.3 Create the Funds JDBC Data source

The JDBC data source defines the connection to the specific database—in this case the Funds database created in section 6.4.1, “Create the Funds database” on page 150.

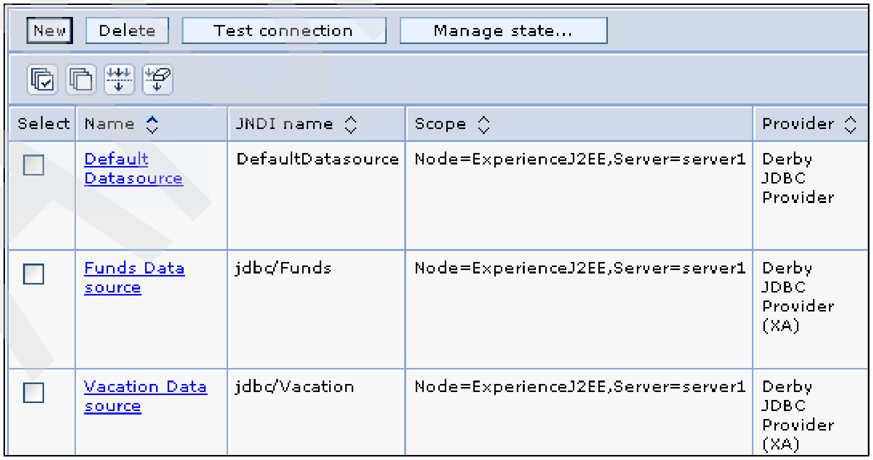
The steps in this section are similar to those used in section 5.4.4, “Create the Vacation JDBC data source” on page 110.

1. From left of the **Admin Console**, select **Resources** → **JDBC** → **Data sources**.
2. On the right at the **Data sources** panel, under **Scope** use the pull-down to change the scope from **All scopes** to **Node=ExperienceJ2EE, Server=server1**.
3. Click **New** to launch the Data sources wizard.
4. At the **Step 1: Enter basic data source information** panel perform the following actions:

- **Data source name**: **Funds Data source**
- **JNDI name**: **jdbc/Funds**
- Under **Component-managed authentication alias and XA recovery authentication alias**, select **ExperienceJ2EE/ExperienceJ2EEAlias** from the pull-down.
- Click **Next**.

5. At the **Step 2: Select JDBC provider** panel perform the following:
   - Select **Select an existing JDBC provider**
   - Select **Derby JDBC Provider (XA)** from the pull-down list
   - Click **Next**.
6. At the **Step 3: Enter database specific properties for the data source** panel perform the following:
   - Windows **Database Name**: **C:\ExperienceJ2EE\Funds**
   - Linux **Database Name**: **/home/*<user>*/ExperienceJ2EE/Funds**

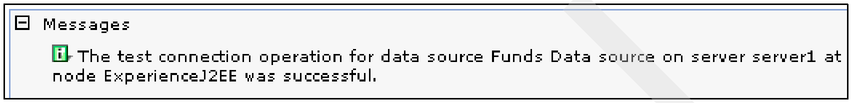
     where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.
   - Ensure that **Use this data source in container managed persistence (CMP)** is selected.
   - Click **Next**.
7. At the **Step 4: Summary panel**, click **Finish**.
8. At the **Data sources** panel, note the new entry.



Select	Name	JNDI name	Scope	Provider
□	Default Datasource	DefaultDatasource	Node=ExperienceJ2EE,Server=server1	Derby JDBC Provider
□	Funds Data source	jdbc/Funds	Node=ExperienceJ2EE,Server=server1	Derby JDBC Provider (XA)
□	Vacation Data source	jdbc/Vacation	Node=ExperienceJ2EE,Server=server1	Derby JDBC Provider (XA)

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

10. Test the JDBC configuration. Return to the **Data sources** window, select the check box next to the **Funds** data source, and click **Test connection**. The test connection should show the following message:



⊟ Messages

The test connection operation for data source Funds Data source on server server1 at node ExperienceJ2EE was successful.

11. Logout and close the **Admin Console**.

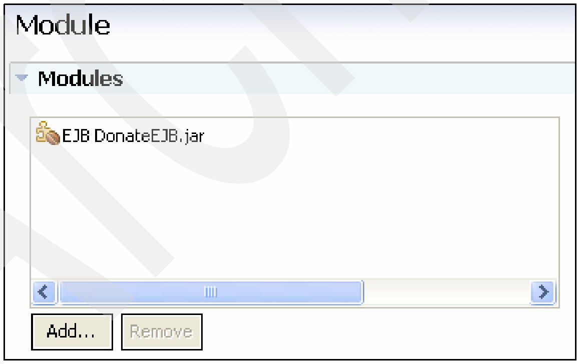
6.6 Add DonateEJB2 to DonateEAR

Now that you made all the configuration changes (and you generated the proper mapping and deployment code), add DonateEJB2 to DonateEAR. This automatically publishes it to the test server.

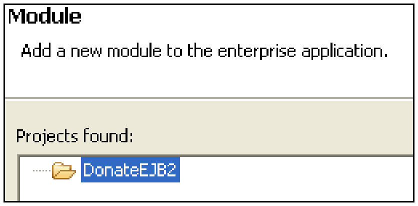
1. From the **Project Explorer**, select **Enterprise Applications** → **DonateEAR** → **DonateEAR**, and double-click to open the deployment descriptor.

2. From the resulting **Application Deployment Descriptor** editor, switch to the **Module** tab and perform the following:

   – Under **Modules** click **Add...**



- At the **Add Module** pop-up, select **DonateEJB2**, and click **Finish**.

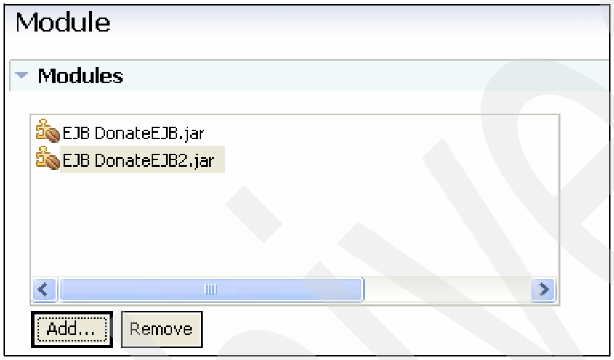


Module

Add a new module to the enterprise application.

Projects found:

3. Back at the **Application Deployment Descriptor** editor, note that it now contains both **DonateEJB** and **DonateDJB2**.



Module

4. Save and close the **Application Deployment Descriptor**.

5. Switch to the **Console** view in the lower right, and look for the messages indicating that the DonateEAR is successfully republished/restarted with DonateEJB2:

```
WSVR0217I: Stopping application: DonateEAR
WSVR0041I: Stopping EJB jar: DonateEJB.jar
WSVR0059I: EJB jar stopped: DonateEJB.jar
WSVR0220I: Application stopped: DonateEAR
WSVR0200I: Starting application: DonateEAR
WSVR0204I: Application: DonateEAR  Application build level: Unknown
WSVR0037I: Starting EJB jar: DonateEJB.jar
WSVR0057I: EJB jar started: DonateEJB.jar
WSVR0037I: Starting EJB jar: DonateEJB2.jar
WSVR0057I: EJB jar started: DonateEJB2.jar
WSVR0221I: Application started: DonateEAR
```

6.7 Test the Funds entity EJB with the UTC

In addition to using the UTC to test the basic function of the Funds entity EJB, you also create the DonationFund record, which we use in the Donate session EJB in Chapter 7, “Create the donate business logic element” on page 165.

6.7.1 Restart the UTC

Restarting the UTC is required to allow the UTC to reference EJBs that were added since it was last initialized:

1. Switch to the **Servers** view in the lower right pane.
2. Select **ExperienceJ2EE Server @ localhost**, and right-click **Restart universal test client**.
3. Click **OK** at the confirmation prompt.
4. Switch to the **Console** view in the lower right pane, and note the following messages (indicating the UTC restarted):

```
WSVR0220I: Application stopped: IBMUTC
WSVR0200I: Starting application: IBMUTC
WSVR0204I: Application: IBMUTC  Application build level: Unknown
SRVE0169I: Loading Web Module: Universal Test Client.
SRVE0250I: Web Module Universal Test Client has been bound to
default_host[*:9080,*:80,*:9443,*:5060,*:5061,*:443].
WSVR0221I: Application started: IBMUTC
```

5. Switch to the **Servers** view in the lower right pane.
6. Select **ExperienceJ2EE Server @ localhost,** and right-click **Run universal test client**. The Universal Test Client opens as a browser window in the upper center pane.
7. From the **Test Client** in the upper left, select **WAS v6 ClassLoading**.
8. From the **WAS v6 ClassLoading** pane (in the right), set **User** to **j2eeadmin** and **Password** to the value you set in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51, and click **Set**. Note that the selection under Modules on the classpath should change from No applications found to DonateEAR .

6.7.2 Test the Funds entity EJB with the UTC

1. From the **Test Client** in the upper left, select **JNDI Explorer**.

2. From the **JNDI Explorer** pane (in the right), locate the reference to the entity EJB local home interface by selecting **[Local EJB Beans]** → **ejb** → **ejbs**. Left-click **FundsLocalHome**.

3. From the **EJB Beans** section (on the left), create the desired entity EJB instance by selecting **EJB Beans** → **FundsLocal** → **FundsLocalHome**.Left-click **create(String)**.

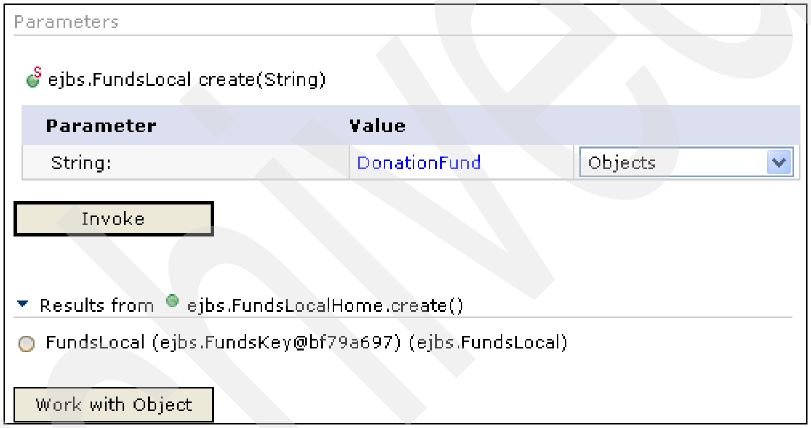
4. From the resulting right pane perform the following:

   – Change the right most selection from **Objects** to **(String)**.

   – **Value**: **DonationFund**

   – Click **Invoke**.

   – After the operation completes, click **Work with object**.



Parameters

ejbs.FundsLocal create(String)

Parameter	Value	
String:	DonationFund	Objects

Results from ejbs.FundsLocalHome.create()

FundsLocal (ejbs.FundsKey@bf79a697) (ejbs.FundsLocal)

Note: In the above image, the right most selection was reset from (String) back to Objects when you clicked invoke.

Behind the scenes:

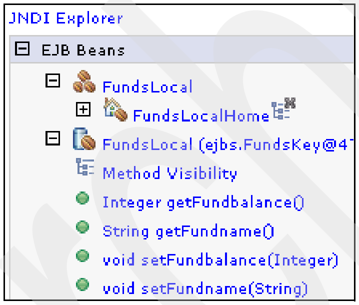
This action creates the following two artifacts:

- A record in the Funds database table with a primary key of DonationFund (and the fundbalance is set to NULL).
- An instance of the FundsLocal entity EJB, which contains the contents of the record.

Since each UTC action is run as a separate transaction, the table record is created immediately (because the transaction is started and committed when you click Invoke).

However, if you were accessing this EJB through user code that manages the transaction (such as a session EJB or a JavaBean that explicitly starts and commits a transaction), the record would not be available to others until the commit operation takes place.

5. Work with the entity EJB instance by expanding on the left to **EJB Bean** → **FundsLocal(FundsKey@xxxxxxx)**.



JNDI Explorer

⊟ EJB Beans

  ⊟ FundsLocal

    ⊞ FundsLocalHome

  ⊟ FundsLocal (ejbs.FundsKey@4

    Method Visibility

    ● Integer getFundbalance()

    ● String getFundname()

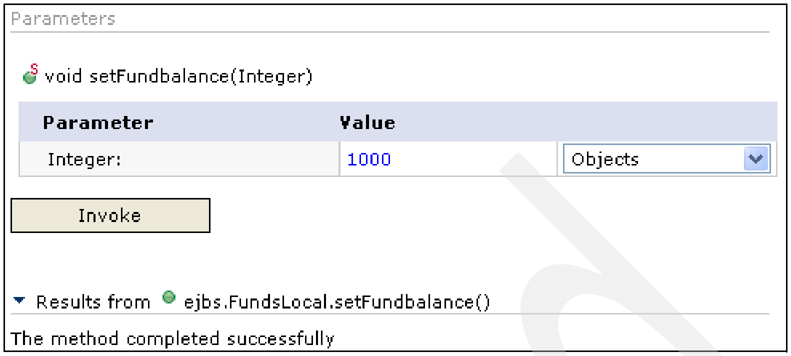
    ● void setFundbalance(Integer)

    ● void setFundname(String)

– Select **void setFundBalance(Integer)** from the left-hand pane, and perform the following from the right-hand pane:

  - Change the right most selection from **Objects** to **0 (Integer)**.
  - **Value**: **1000**
  - Click **Invoke**.

- View the result in the bottom half of the right-hand pane.



Parameters

void setFundbalance(Integer)

Parameter	Value	
Integer:	1000	Objects

Results from ejbs.FundsLocal.setFundbalance()

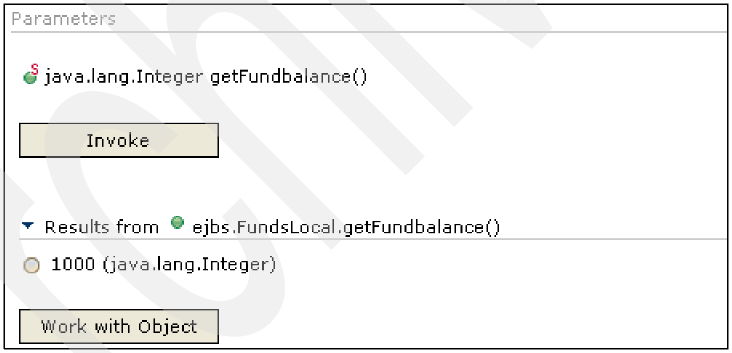
The method completed successfully

Note: In the above image, the right most selection was reset from 0 (Integer) back to Objects when you clicked invoke.

– Select **Integer getFundBalance()** from the left-hand pane.

– From the right-hand pane perform the following:

- Click **Invoke**.
- Verify that the Fund balance was set to the value entered in step 9 (1000).



Parameters

java.lang.Integer getFundbalance()

Results from ejbs.FundsLocal.getFundbalance()

1000 (java.lang.Integer)

6. Close the Test Client.

Extra credit:

If you want to verify that the update was successfully made to the database, you must first stop **ExperienceJ2EE Server @localhost**, because Derby only allows a single Java process to access the database. Following are the basic steps:

1. From the **Servers** view in the lower-right pane, select **ExperienceJ2EE Server @localhost**.
2. Right-click **Stop**.
3. From the **Database Explorer** view in the lower-right pane perform the following:
   – Select **Connections** → **Funds**, and right-click **Reconnect**.
      - At the **Database Authorization** pop-up click **OK** (no password is required).

   – Select **Connections** → **Funds[Derby 10.1]** → **Funds** → **Schemas** → **EXPERIENCE** → **FUNDS**, and right-click **Data** → **Sample Contents**.

   At the resulting **Data Output** view in the lower-right pane, note the new entry for DonationFund with a fund balance of 1000 (assuming that you made the changes outlined above).
4. Switch back to the **Database Explorer** view, select **Connections** → **Funds[Derby 10.1]**, and right-click **Disconnect**.
5. From the **Servers** view in the lower-right pane, select **ExperienceJ2EE Server @localhost**, and right-click **Start**.
6. Switch to the **Console** view in the lower-right pane, and ensure that the server starts without errors.

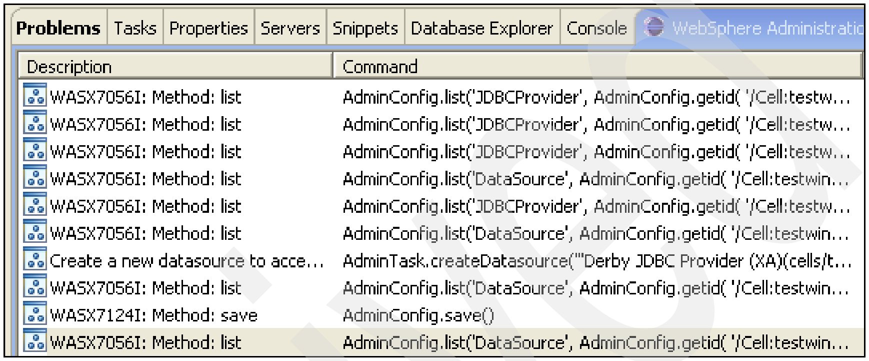
If the Database Explorer view does not appear or if the Funds database is not in the list of connections, repeat the steps in section 6.4.1, “Create the Funds database” on page 150.

6.8 Explore!

This IBM Redbook describes how to create various WAS V6.1 configuration settings using the Admin Console. This is fine for basic unit testing, but for integration testing and deployment you probably want to have an automated way of making configuration changes. The challenge is how do you accurately convert from the Admin Console steps to an automation script?

AST V6.1 introduces a new feature called WebSphere administration command assist that helps in this task.

You can open the command assist from the **Servers** view (select the server and right-click → **WebSphere administration command assist)**. Once this is open, it monitors the interaction between the Admin Console and the test server and then displays the command script (jython) equivalent of the Admin console action. Jython is the primary scripting language supported for WAS V6.1 configuration scripts.



Description	Command
WASX7056I: Method: list	AdminConfig.list('JDBCProvider', AdminConfig.getid('/Cell:testw...
WASX7056I: Method: list	AdminConfig.list('JDBCProvider', AdminConfig.getid('/Cell:testw...
WASX7056I: Method: list	AdminConfig.list('JDBCProvider', AdminConfig.getid('/Cell:testw...
WASX7056I: Method: list	AdminConfig.list('DataSource', AdminConfig.getid('/Cell:testwin...
WASX7056I: Method: list	AdminConfig.list('JDBCProvider', AdminConfig.getid('/Cell:testw...
WASX7056I: Method: list	AdminConfig.list('DataSource', AdminConfig.getid('/Cell:testwin...
Create a new datasource to acce...	AdminTask.createDatasource('"Derby JDBC Provider (XA)(cells/t...
WASX7056I: Method: list	AdminConfig.list('DataSource', AdminConfig.getid('/Cell:testwin...
WASX7124I: Method: save	AdminConfig.save()
WASX7056I: Method: list	AdminConfig.list('DataSource', AdminConfig.getid('/Cell:testwin...

AST V6.1 supports a special project type (Jython) and artifact type (Jython script file) that develop command scripts. The usage of these with the command assist feature is straight forward:

1. Open both the command assist and the Jython script editor.
2. From the Admin Console, execute the desired configuration commands.
3. From the **WebSphere Administration Command** view, select the desired change, and right-click → **Insert**. This copies the command to the current line in the Jython script.
4. Test the script by selecting it in the **Project Explorer** view, and right-clicking **Run As** → **Administrative Script**.

Additional explore resources are available at the following Web sites:

- Jython project:

  http://www.jython.org

- AST V6.1 InfoCenter: Developing Automation Scripts:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.ws.ast.jythontools.doc/topics/cautoscript.html

7

Create the donate business logic element

In this chapter you create a new session EJB that contains donateVacation, which is the business logic of your new application. It also transfers vacation from an Employee record to a Funds record.

7.1 Learn!

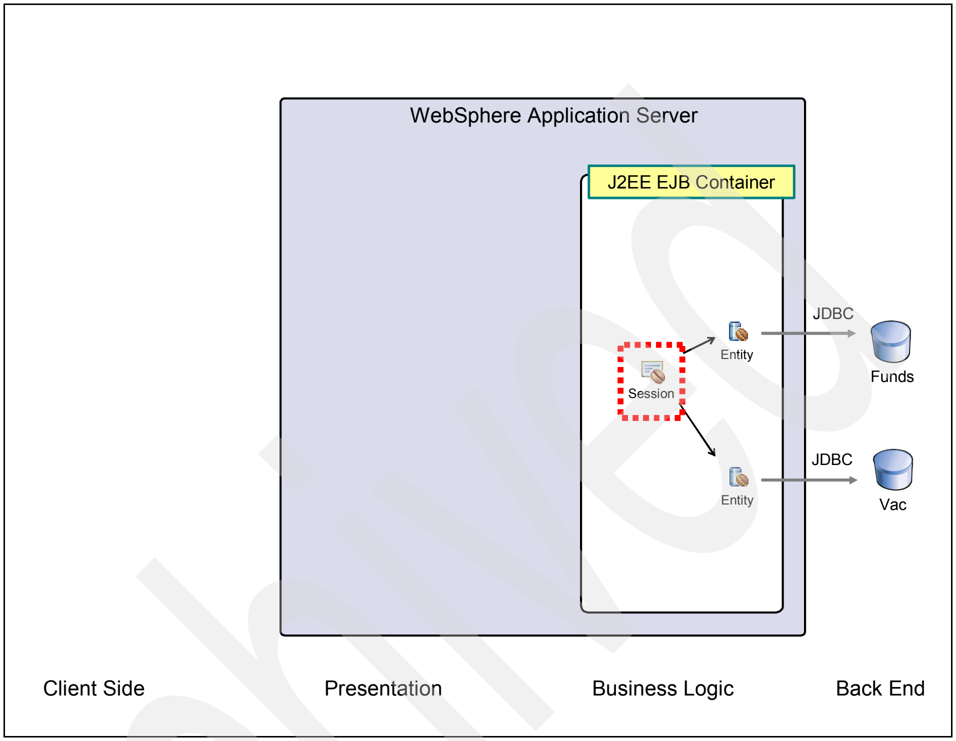


Figure 7-1 Donate application: Donate session EJB

Entity EJBs, like those you created in the preceding chapters, encapsulate data into a record-oriented format. The next challenge is how do you link these together to form a business function that acts against one or more of these entity EJBs?

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 the same local/remote invocation mechanisms used with entity EJBs. But then the container invokes the user-written session EJB method instead of the find, create, or remove functions of the entity EJBs.

The session EJB method is run as a single transaction, and when the method completes all actions against the various entity EJBs (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.

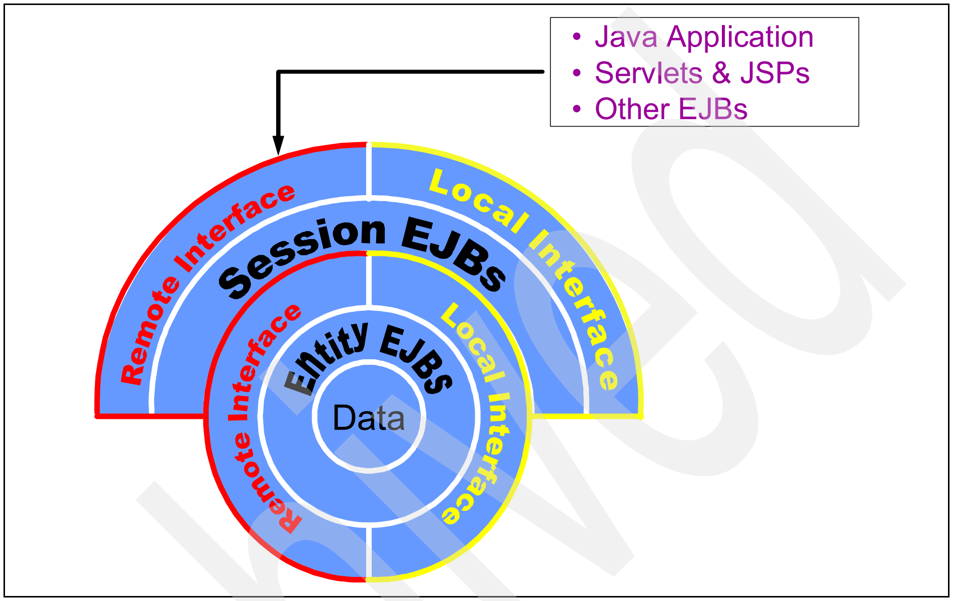


Figure 7-2 Session EJBs

As shown in Figure 7-2, the typical usage pattern is for client programs to access session EJBs using the remote interface. This provides the fail-over and workload management that exist when using a J2EE application server in a cluster environment. In this redbook, you are using a single-instance version of WAS V6.1, so failover and workload management are not applicable.

Next, the session EJBs, through the user-written code, accesses the associated entity EJBs through the local interface. This provides much higher performance than accessing them through the remote interface, and it matches the typical deployment pattern used in most J2EE deployments, where session EJBs and entity EJBs are located on the same system and the same JVM.

The downside to this approach is that you must deploy the session EJBs and entity EJBs together, and you lose the fail-over and workload management capabilities that might exist between them. Note that you still have the fail-over and workload management capabilities between the client program and the session EJB.

Most experienced J2EE developers believe this is a reasonable compromise—that the improved performance far outweighs the drawbacks.

This usage pattern is just that, *A usage pattern*. You can still access entity EJBs directly through the remote interface or session EJBs through the local interface: All combinations are allowed by the J2EE specifications.

Session EJBs can also be used in situations with no entity EJBs 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:

- Local/Remote invocation, along with the associated failover and workload management.
- Common access and security control across all technologies (Web front end, J2EE 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 may 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 needs to maintain state or “conversational context” across session EJB invocations, an alternate design pattern is to store the ‘state’ information in an entity EJB 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 EJB.

Additional learn resources are available at the following Web sites:

- EJB 2.1 Final Release:

  `http://java.sun.com/products/ejb/docs.html`
- The J2EE 1.4 Tutorial: Chapter 25: Session Bean Examples:

  `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

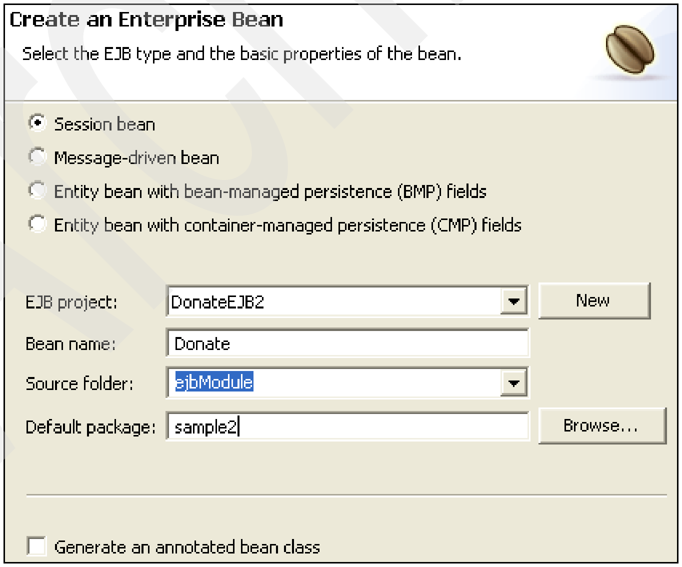
- IBM developerWorks: The top 10 (more or less) J2EE best practices: Use stateless session beans instead of stateful session beans:

  http://www.ibm.com/developerworks/websphere/techjournal/0405_brown/0405_brown.html#sec7

7.2 Define the Donate session EJB

In this section you define the session EJB in an EJB project. This section uses the DonateEJB2 project, but you can also use DonateEJB.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Session Beans**, and right-click **New** → **Session Bean**.
2. At the **Create an Enterprise Bean** pop-up populate the following fields:
   - Select **Session bean**.
   - **EJB Project**: **DonateEJB2**
   - **Bean name**: **Donate**
   - **Source folder**: **ejbModule**
   - **Default package**: **sample2**
   - IMPORTANT! Ensure that **Generate an annotated bean class** is cleared. See Behind the scenes below.
   - Click **Next**.



Create an Enterprise Bean

Select the EJB type and the basic properties of the bean.

● Session bean

○ Message-driven bean

○ Entity bean with bean-managed persistence (BMP) fields

○ Entity bean with container-managed persistence (CMP) fields

EJB project: DonateEJB2

Bean name: Donate

Source folder: ejbModule

Default package: sample2

☐ Generate an annotated bean class

Behind the scenes:

For this book, you must clear **Generate an annotated bean class** to allow you to add the method-level permission security to the Donate session EJB using the EJB deployment descriptor as described in section 11.4.2, “Restrict access to Donate session EJB via declarative security” on page 270.

Otherwise, if you created this session EJB as an annotated bean class you would have to add the method-level permission using the @ejb.permission annotation tag.

3. At the **Enterprise Bean Details** pop-up, review the default values that define a **Remote client view** only (you should not have to make any changes on this pop-up), and click **Finish**.

   Note that you should initially see one warning in the **Problems** view. After the project is automatically published, you will see additional warnings of the same type. We fix this in the next section.

   ```
   The serializable class DonateBean does not declare a static final
   serialVersionUID field of type long
   ```

7.2.1 Alter the workspace settings to ignore warnings

Warning(s) appear in the **Problem** view (in the lower-right pane) after defining the session EJB. Additional warnings appear after the EJB deploy code is updated as part of the automatic Publishing to the test server. All of these warnings indicate that a serial Version ID variable is not defined.

```
The serializable class DonateBean does not declare a static final
serialVersionUID field of type long
```

These warnings can be ignored because defining a serialVersionUID is not required. It is a “good programming practice” if you intend to have different versions of the same class (and desire a way to distinguish between which version of the class you are working with), but in this scenario you will always have a single version.

Some circles consider that adding this optimizes performance, but realistically this would have at best a very small impact the first time the class is loaded because the JVM caches the computed result for further access to the same class.

1. From the AST V6.1 **J2EE Perspective** action bar, select **Window** → **Preferences...**

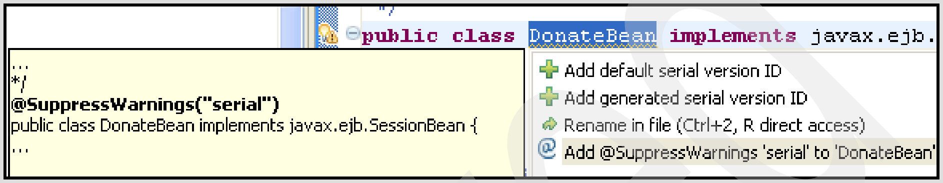
2. At the **Preferences** pop-up perform the following:
   - On the left, select **Java** → **Compiler** → **Errors/Warnings**.
   - On the right (at the resulting **Errors/Warnings** pane), expand the **Potential programming problems** section, and change the option for S**erializable class without serialVersionUID** from **Warning** to **Ignore**.
   - Click **OK**.
   - At the resulting **Error/Warnings Settings Changed** pop-up, select **Yes**.

   The **Problems** view should now show no errors or warnings.

Alternative:

You could disable the specific occurrence of this warning, instead of disabling this warning globally for all occurrences in all projects in the workspace:

- From the **Problems** view, double-click the warning. The Java editor opens automatically and positions you at the line containing the warning.
- In the editor, left-click on the quick fix icon [💡] in the left-hand margin, and select **Add @SuppressWarnings...** from the list.



```
public class DonateBean implements javax.ejb.
```

```
...
*/
@SuppressWarnings("serial")
public class DonateBean implements javax.ejb.SessionBean {
...
```

Add default serial version ID
Add generated serial version ID
Rename in file (Ctrl+2, R direct access)
Add @SuppressWarnings 'serial' to 'DonateBean'

This action adds the selected line to the file, suppressing this type of warning only in this Java file:

```
/**
 * Bean implementation class for Enterprise Bean: Donate
 */
@SuppressWarnings("serial")
public class DonateBean implements javax.ejb.SessionBean {
```

Which option is better, fixing these warnings globally or individually? Ideally, you probably want to fix these individually because the default compiler settings are intended to represent “good programming practices”. Therefore, it is better to alter your code to match the recommended programming practices rather than the other way around.

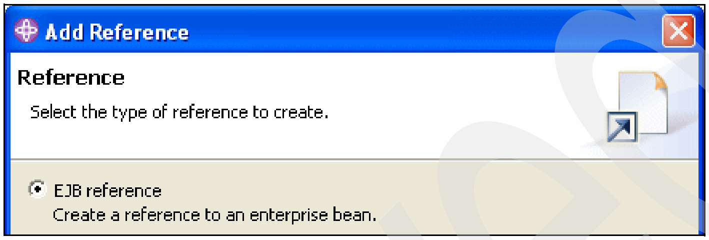
However, in this case, many of these warnings are contained in automatically generated code, the EJB deployment code. The individual overrides are wiped out each time you regenerate the deployment code.

Therefore, the approach used in this book is to alter the compiler errors/warnings settings.

7.2.2 Create the EJB references

The donateMethod that you will create in the next section accesses the Employee and Funds entity EJBs that you developed in previous chapters. Per J2EE best practices, you access these by using EJB references (instead of a direct JNDI lookup).

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2**, and double-click to open the deployment descriptor.
2. From the resulting **EJB Deployment Descriptor** editor, switch to the **References** tab.
3. Add the Employee EJB reference using the following tasks:
   - Under References, select **Donate**, and click **Add...**
   - At the resulting **Reference** pop-up, select **EJB reference**, and click **Next**.

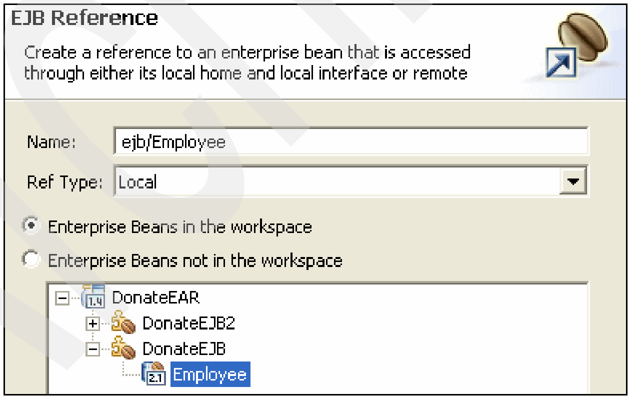


Reference

Select the type of reference to create.

● EJB reference
Create a reference to an enterprise bean.

   - At the **Add EJB Reference** pop-up, perform the following actions:
     - Select **Enterprise Beans in the workspace**.
     - Select **DonateEAR** → **DonateEJB** → **Employee.** Note that the **Name** field is automatically populated with **ejb/Employee**.
     - Set **Ref Type** to **Local**.
     - Click **Finish**.



EJB Reference

Create a reference to an enterprise bean that is accessed through either its local home and local interface or remote

Name: ejb/Employee

Ref Type: Local

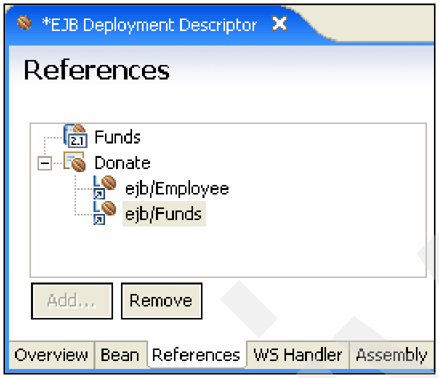
● Enterprise Beans in the workspace

○ Enterprise Beans not in the workspace

4. Add the Funds EJB reference by performing the following actions:
   - Under References, select **Donate**, and click **Add...**
   - At the resulting **Reference** pop-up, select **EJB reference**, and click **Next**.
   - At the **Add EJB Reference** pop-up, perform the following tasks:

   - Select **Enterprise Beans in the workspace**.
   - Select **DonateEAR** → **DonateEJB2** → **Funds**. Note that the **Name** field is automatically populated with **ejb/Funds**.
   - Set **Ref Type** to **Local**.
   - Click **Finish**.

5. When finished, the reference screen should appear similar to the following:



6. Save and close the **EJB deployment descriptor**.

Behind the scenes:

The local EJB references are stored in the EJB deployment descriptor (**EJB Modules** → **DonateEJB2** → **ejbModule-META-INF** → **ejb-jar.xml**). The following shows the stanza for the Employee EJB reference:

```
<ejb-local-ref id="EJBLocalRef_1152104289080">
   <description>
   </description>
   <ejb-ref-name>ejb/Employee</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <local-home>sample.EmployeeLocalHome</local-home>
   <local>sample.EmployeeLocal</local>
   <ejb-link>DonateEJB.jar#Employee</ejb-link>
</ejb-local-ref>
```

References allow J2EE artifacts to identify and access the Java Naming and Directory Interface™ (JNDI) name of J2EE components through a reference (alias) that then points to the actual JNDI name.

This redirection allows J2EE administrators to reconfigure applications at deployment time in order to meet site specific naming standards or to resolve naming conflicts.

Your application accesses entity EJBs called ejb/Employee and ejb/Funds. What if someone else used the same names?

J2EE Administrators can override the JNDI names for the entity EJBs when deploying the applications, but what about the application code that calls the renamed entity EJBs?

- Without references, one of the applications would have to be recoded to use the new JNDI name.
- With references, the J2EE Administrator can reconfigure the reference when deploying the application.

J2EE does not require you to use references, but it is considered to be a very important "Best Practice."

IBM has gone one step further and deprecated the direct lookup function (indicating that although it still exists in this release, it may be removed from future releases). The following quote comes from the WebSphere Application Server InfoCenter:

> *“Though you can use a direct JNDI name, this naming method is deprecated in Version 6 of WebSphere Application Server. Application Server assigns default values to the resource-reference data when you use this method. An informational message, resembling the following, is logged to document the defaults:*

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

...”

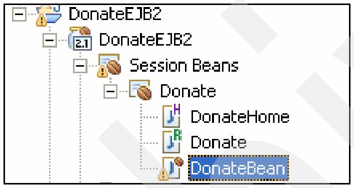
7.3 Code the Donate session EJB

In this section you add your business logic to the generated session EJB, which at this point contains only the infrastructure code needed to initialize the overall bean:

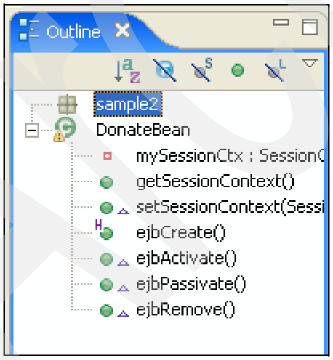
- Create the initial donateVacation method using a snippet created in section 3.8, “Import the sample code snippets” on page 64.
- Complete the donationVacation method using EJB snippets (shipped with AST V6.1) to insert the calls to the Funds and Employee entity EJBs.

7.3.1 Create the donateVacation method

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Session Beans** → **Donate** → **DonateBean**(.java)**,** and double-click to open the Java class.

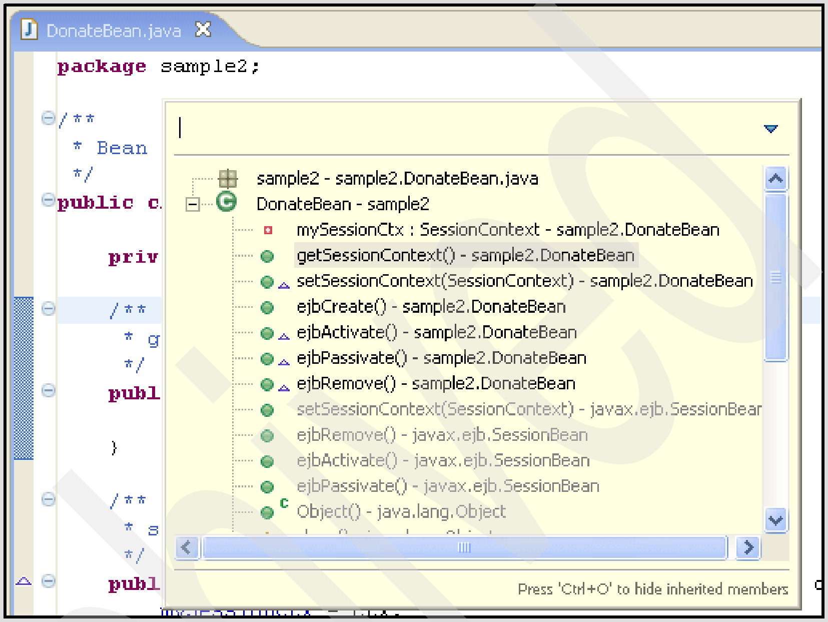


Note that the Java editor opens in the upper center pane, and a Java outline opens in the upper-right pane—using the Java outline makes it easier to switch to the various sections of the Java class.



Alternative:

AST V6.1 also supports a variation of the outline called the In-place outline. This can be launched from the editor by clicking **Ctrl+O** or from the action bar by selecting **Navigate** → **Quick Outline**.



DonateBean.java

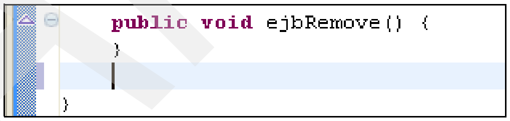
```
package sample2;

/**
 * Bean
 */
public c

    priv

    /**
     * g
     */
    publ

    }

    /**
     * s
     */
    publ
```

sample2 - sample2.DonateBean.java
DonateBean - sample2
mySessionCtx : SessionContext - sample2.DonateBean
getSessionContext() - sample2.DonateBean
setSessionContext(SessionContext) - sample2.DonateBean
ejbCreate() - sample2.DonateBean
ejbActivate() - sample2.DonateBean
ejbPassivate() - sample2.DonateBean
ejbRemove() - sample2.DonateBean
setSessionContext(SessionContext) - javax.ejb.SessionBear
ejbRemove() - javax.ejb.SessionBean
ejbActivate() - javax.ejb.SessionBean
ejbPassivate() - javax.ejb.SessionBean
Object() - java.lang.Object

Press 'Ctrl+O' to hide inherited members

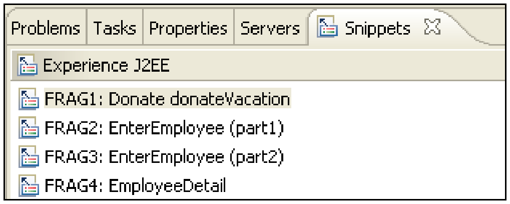
The value of this form versus the standard **Outline** view is that it allows you to show inherited members. In this case, you can see the inherited EJB methods such as ejbRemove and ejbActivate.

2. In the **DonateBean.java** editor, insert a blank line before closing the bracket, and position the cursor on this new line.



```
        public void ejbRemove() {
        }
        
}
```

3. In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG1: Donate donateVacation**. This inserts the snippet code into **DonateBean.java** at the current cursor location.


Problems
Tasks
Properties
Servers
Snippets
Experience J2EE
FRAG1: Donate donateVacation
FRAG2: EnterEmployee (part1)
FRAG3: EnterEmployee (part2)
FRAG4: EmployeeDetail

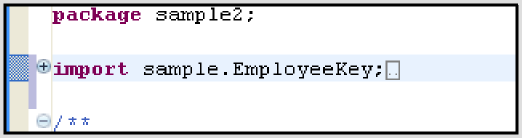
```
public String donateVacation(int employee_num, String fund_name,
         int donation_amount) {

   String return_string = "Default Message: error on transfer";
   EmployeeKey primaryKey = new EmployeeKey(new java.lang.Integer(
         employee_num));
   FundsKey fund_key = new FundsKey(fund_name);
   // * Insert call to Employee Entity Bean
   if (anEmployeeLocal == null) {
      return_string = "Error: Employee Record " + employee_num
            + " not found";
   } else {
      // Insert call to Funds Entity Bean
      if (aFundsLocal == null) {
         return_string = "Error: Fund Record " + fund_name + " not found";
      } else {
         try {
            anEmployeeLocal.setVacamount(
               anEmployeeLocal.getVacamount() - donation_amount);
            aFundsLocal.setFundbalance(
               aFundsLocal.getFundbalance() + donation_amount);
            return_string = "Transfer Successful";
         } catch (Exception e) {
            return_string = "Error on transfer: "+ e.getLocalizedMessage();
         }
      }
   }
   return return_string;
}
```

Example 7-1 FRAG1: Donate donateVacation

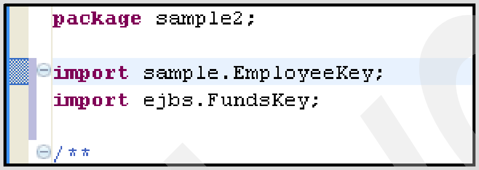
4. Organize the imports by selecting anywhere on the editor, and right-clicking **Source** → **Organize imports** (or **Ctrl+Shift+O,** or select **Source** → **Organize Imports** from the action bar) to remove most of the errors. Some errors will remain until you add the calls to the Employee and Funds entity EJBs.

Behind the scenes:

Organizing imports adds the specified classes to the list of imports at the top of the Java file. Note that the list of imports may be collapsed or expanded by clicking the plus (+) symbol on the left or clicking enter on the... box on the right.



```
package sample2;

import sample.EmployeeKey;

/**
```



```
package sample2;

import sample.EmployeeKey;
import ejbs.FundsKey;

/**
```

You can determine the location of the class, either in one of your projects or in one of the WAS V6.1 jar files, by selecting the import, and then right-clicking **Declarations** → **Workspace**.

This is actually quite useful, since the jar files that are accessible in your workspace (which are defined in the project properties classpath) are different than those available at runtime (which are controlled by a combination of the overall classpath of the application server combined with the jar files defined in the project's MANIFEST.MF file). A mismatch between these two environments can result in a NoClassDefFoundError exception.

The problem described in section 7.3.5, "Work around: Add serviceLocatorMgr.jar to MANIFEST.MF" on page 186 is a good example of a situation where the development classpath is configured properly and the runtime classpath is not.

5. To format the code, select anywhere on the editors and right-click **Source** → **Format**—or **Ctrl+Shift+F** on Windows (**Esc** and then **Ctrl-F** on Linux)**,** or select **Source** → **Format** from the action bar).

Behind the scenes:

- A default return message is created:

```
String return_string = "Default Message: error on transfer";
```

- The native input values (employee_num and fund_name) are used to create the respective EJB key objects (EmployeeKey and FundsKey, respectively):

```
EmployeeKey primaryKey = new EmployeeKey(new java.lang.Integer(
            employee_num));
FundsKey fund_key = new FundsKey(fund_name);
```

- The Employee entity EJB interface is called (TO BE CODED – will be done in subsequent steps) to locate and return an instance of Employee that matches this key, and if not found, an error message is generated:

```
//* Insert call to Employee Entity Bean

if (anEmployeeLocal == null) {
   return_string = "Error: Employee Record " + employee_num + "
not found";
} else {
```

  - The Funds entity EJB interface is called (TO BE CODED – will be done in subsequent steps) to locate (and return) an instance of Funds that matches the input fund (and if not found, an error message is generated):

```
// Insert call to Funds Entity Bean


   if (aFundsLocal == null) {
   return_string = "Error: Fund Record " + fund_name + " not
found";
} else {
```

- The donation amount is subtracted from the Employee instance:

```
anEmployeeLocal.setVacamount(
   anEmployeeLocal.getVacamount() - donation_amount);
```

- The donation amount is added to the Fund instance:

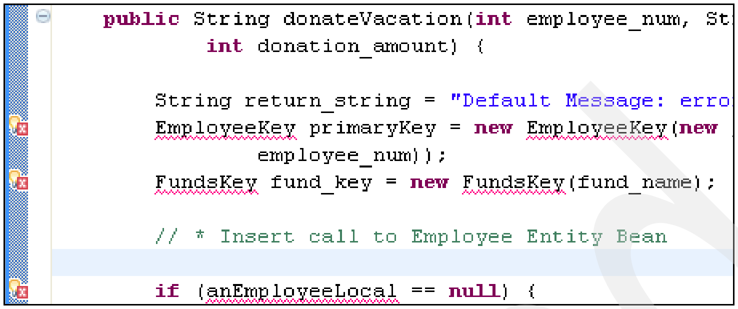
```
aFundsLocal.setFundbalance(
      aFundsLocal.getFundbalance() + donation_amount);
```

- If no errors were encountered, a successful return message is set:

```
return_string = "Transfer Successful";
```

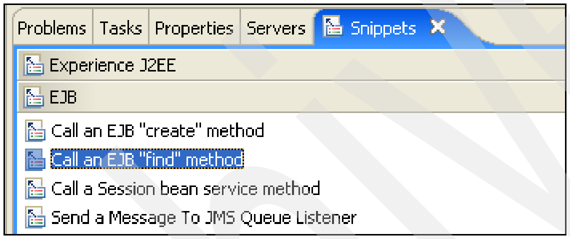
7.3.2 Add the call to the Employee entity EJB using a snippet

1. Select the line after **//* Insert call to Employee Entity Bean**.



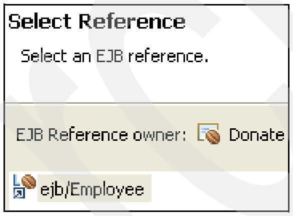
```
public String donateVacation(int employee_num, St
        int donation_amount) {

    String return_string = "Default Message: erro
    EmployeeKey primaryKey = new EmployeeKey(new 
            employee_num));
    FundsKey fund_key = new FundsKey(fund_name);

    // * Insert call to Employee Entity Bean

    if (anEmployeeLocal == null) {
```

2. In the **Snippets** view (lower right), expand the **EJB** category, and double-click **Call an EJB “find” method**.



Problems
Tasks
Properties
Servers
Snippets
Experience J2EE
EJB
Call an EJB "create" method
Call an EJB "find" method
Call a Session bean service method
Send a Message To JMS Queue Listener

3. At the **Select Reference** pop-up, select **ejb/Employee**, and click **Next**.



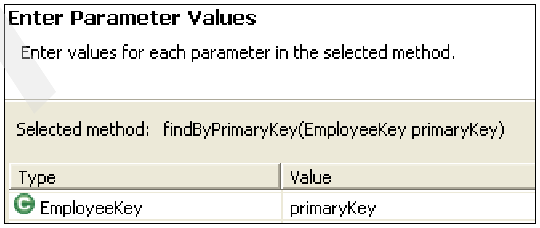
Select Reference

Select an EJB reference.

EJB Reference owner: Donate

ejb/Employee

4. At the **Enter Parameter Values** pop-up, note that the **primaryKey** value matches the name of the EmployeeKey object used in the donateVacation method, which you inserted using the snippet. Click **Finish**.



Enter Parameter Values

Enter values for each parameter in the selected method.

Selected method: findByPrimaryKey(EmployeeKey primaryKey)

Type	Value
EmployeeKey	primaryKey

Behind the scenes:

This snippet made the following changes in DonateBean.java:

- Added the following to define the EJB reference name and class names for the entity EJB:

```
private final static String STATIC_EmployeeLocalHome_REF_NAME =
   "ejb/Employee";
private final static Class STATIC_EmployeeLocalHome_CLASS =
   EmployeeLocalHome.class;
```

- Added the following method to locate the record. Refer below for a discussion on ServiceLocatorManager.

```
protected EmployeeLocal find_EmployeeLocalHome_findByPrimaryKey(
   EmployeeKey primaryKey)
      EmployeeLocalHome anEmployeeLocalHome = (EmployeeLocalHome)
         ServiceLocatorManager..getLocalHome(
            STATIC_EmployeeLocalHome_REF_NAME,
            STATIC_EmployeeLocalHome_CLASS);
   try {
      if (anEmployeeLocalHome != null)
         return anEmployeeLocalHome.findByPrimaryKey(primaryKey);
   } catch (javax.ejb.FinderException fe) {
      // TODO Auto-generated catch block
      fe.printStackTrace();
   }
   return null;
}
```

- In the donateVacation method, added the selected line of code to call the find_EmployeeLocalHome method to locate the record:

```
// * Insert call to Employee Entity Bean
EmployeeLocal anEmployeeLocal =
   find_EmployeeLocalHome_findByPrimaryKey(primaryKey);
```

Note that find_EmployeeLocalHome_findByPrimaryKey uses the ServiceLocatorManager.getLocalHome method to lookup the Employee entity EJB in the name space:

```
EmployeeLocalHome anEmployeeLocalHome = (EmployeeLocalHome)
   ServiceLocatorManager.getLocalHome(
      STATIC_EmployeeLocalHome_REF_NAME,
      STATIC_EmployeeLocalHome_CLASS);
```

ServiceLocatorManager is an IBM specific implementation class that is used instead of the standard J2EE JNDI lookup. In this, a standard JNDI lookup replaces the above line with the following:

```
InitialContext context;
EmployeeLocalHome anEmployeeLocalHome = null;
try {
   context = new InitialContext();
   anEmployeeLocalHome = (EmployeeLocalHome)
      context.lookup("java:comp/env/ejb/Employee");
} catch (Exception e) {
      System.out.println("Exception on looking up JNDI name: " +
         e.getLocalizedMessage());
}
```

Why the IBM specific implementation?

- ServiceLocatorManager caches JNDI lookups (whereas the standard J2EE mechanism does not). This provides improved runtime performance.
- ServiceLocatorManager does not throw exceptions, whereas the standard J2EE mechanism does. This leads to simplified user code because you simply check for a null return instead of catching various exceptions. Note that if you want to handle exceptions, you can register a special ServiceLocatorErrorHandler interface.
- ServiceLocatorManager only uses JNDI references, thus enforcing a best practice of only accessing JNDI names via references and automatically prefixes the JNDI reference string (“java:comp/env”).

  The standard J2EE mechanism will use either the direct local lookup:

  ```
  "local:ejb/ejb/sample/EmployeeLocalHome"
  ```

  or the prefixed JNDI reference:

  ```
  "java:comp/env/ejb/Employee"
  ```

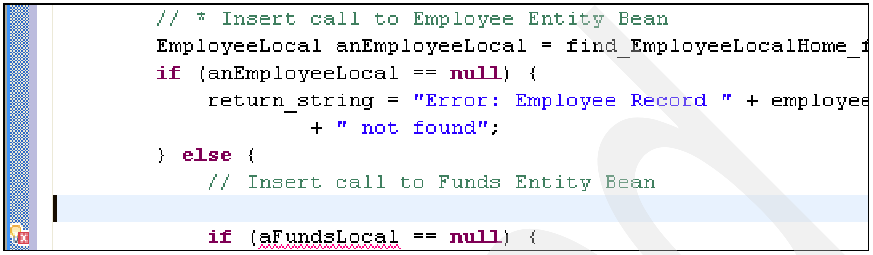
  where the ServiceLocatorManager uses the unprefixed JNDI reference:

  ```
  "ejb/Employee"
  ```

7.3.3 Add the call to the Funds entity EJB using a snippet

The steps used to add the call the Funds entity EJB are similar to those used for the Employee entity EJB.

1. Select the line after **//* Insert call to Funds Entity Bean**.



```
        // * Insert call to Employee Entity Bean
        EmployeeLocal anEmployeeLocal = find_EmployeeLocalHome_f
        if (anEmployeeLocal == null) {
            return_string = "Error: Employee Record " + employee
                    + " not found";
        } else {
            // Insert call to Funds Entity Bean

            if (aFundsLocal == null) {
```

2. In the **Snippets** view (lower right), expand the **EJB** category, and double-click **Call an EJB “find” method**.
3. At the **Select Reference** pop-up, select **ejb/Funds**, and click **Next**.
4. At the **Enter Parameter Values** pop-up, change the **Value** field from **primaryKey** to **fund_key** (to match the name of the FundsKey object used in the donateVacation that you inserted using the snippet), and click **Finish**.

Behind the scenes:

Recall that you used the default key parameter name primaryKey when you inserted the find method for the Employee object, and this matched the EmployeeKey primaryKey variable defined in the snippet.

Therefore, you need to use a different default key parameter name when inserting this snippet for the Funds object in order to match the FundsKey fund_key variable defined in the snippet.

If you did not change this value, AST V6.1 would display the following error:

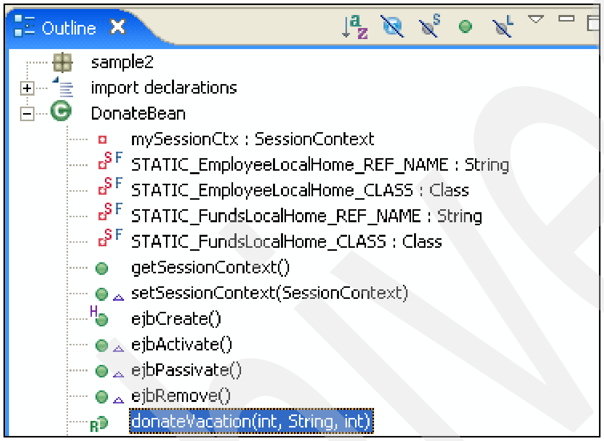
```
The method find_FundsLocalHome_findByPrimaryKey(FundsKey) in
the type DonateBean is not applicable for the arguments
(EmployeeKey)
```

5. Save **DonateBean.java**. All errors should be resolved in the Problems pane. DO NOT CLOSE.

7.3.4 Complete the session EJB

The user method is public but not externally available through the remote interface until it is promoted, which adds a declaration of the method to the remote interface file (in this case, Donate.java).

1. Switch to the **Outline** view in the upper- right pane.
2. Select the **donateVacation** method, and right-click **Enterprise Bean** → **Promote to Remote Interface**. The letter ‘R’ is added next to this method, indicating that it is available through the remote interface.



```
sample2
import declarations
DonateBean
    mySessionCtx : SessionContext
    STATIC_EmployeeLocalHome_REF_NAME : String
    STATIC_EmployeeLocalHome_CLASS : Class
    STATIC_FundsLocalHome_REF_NAME : String
    STATIC_FundsLocalHome_CLASS : Class
    getSessionContext()
    setSessionContext(SessionContext)
    ejbCreate()
    ejbActivate()
    ejbPassivate()
    ejbRemove()
    donateVacation(int, String, int)
```

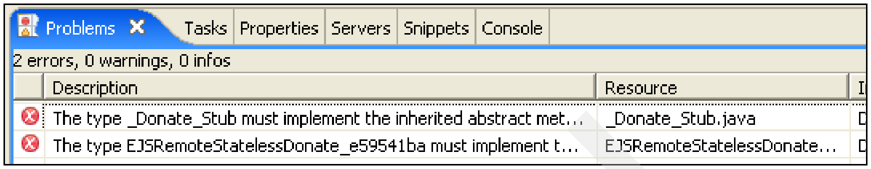
Work around:

The **Outline** view may not update with the letter ‘R’ after promoting the donateVacation method to the remote interface. If this happens, close the **DonateBean.java** editor and reopen. The Outline view should be reloaded correctly.

This is a known problem that is fixed in Web Tools Platform 1.5 and will be included in the AST V6.1 update/version that includes WTP 1.5.

3. Save and close **DonateBean.java**.

4. The **Problems** view initially shows the following error messages, indicating that the EJB deployment code is out of sync with the updated session EJB.



2 errors, 0 warnings, 0 infos

	Description	Resource
⊗	The type _Donate_Stub must implement the inherited abstract met...	_Donate_Stub.java
⊗	The type EJSRemoteStatelessDonate_e59541ba must implement t...	EJSRemoteStatelessDonate...

The automatic publishing process should regenerate the EJB deployment code in the background and these errors should disappear.

Off course?

If the errors do not completely disappear within approximately a minute, the server may not have detected the latest changes.

You should be able to resolve this by forcing a re-publication:

- Switch to the **Servers** view, select **ExperienceJ2EE Server @ localhost**, and right-click **Publish**.

The publishing process re-runs, and the errors should then disappear.

7.3.5 Work around: Add serviceLocatorMgr.jar to MANIFEST.MF

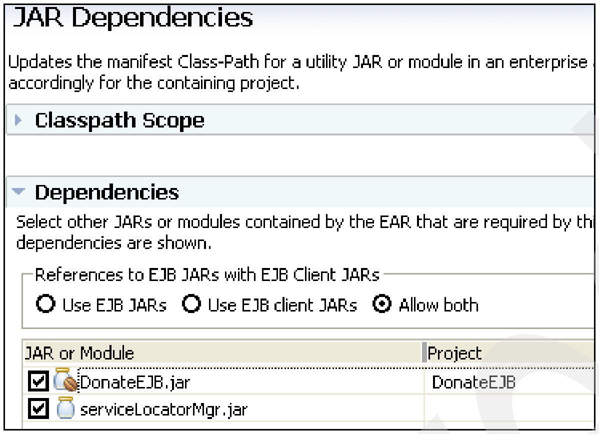
This jar file (contained in DonateEAR) contains classes that are used by the EJB find method code added by the snippet. In Rational Application Developer V6.0, this was automatically added to both the classpath and the MANIFEST.MF of the EJB project when running the snippet. In AST V6.1 it is only added to the classpath (and not to the MANIFEST.MF). This is a known problem that is fixed in Web Tools Platform 1.5 and will be included in the AST V6.1 update/version that includes WTP 1.5.

As a result the donateMethod fails with the following error when invoking the snippet code:

```
CNTR0020E: EJB threw an unexpected (non-declared) exception during
invocation of method "donateVacation" on bean
"BeanId(DonateEAR#DonateEJB2.jar#Donate, null)". Exception data:
java.lang.NoClassDefFoundError:
com.ibm.etools.service.locator.ServiceLocatorManager
```

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **ejbModule** → **META-INF** → **MANIFEST.MF**, and double-click to open.

2. From the resulting **Jar Dependencies** editor, scroll down to the **Dependencies** section, and select **serviceLocatorMgr.jar**.



JAR Dependencies

Updates the manifest Class-Path for a utility JAR or module in an enterprise accordingly for the containing project.

Classpath Scope

Dependencies

Select other JARs or modules contained by the EAR that are required by th dependencies are shown.

References to EJB JARs with EJB Client JARs

○ Use EJB JARs ○ Use EJB client JARs ● Allow both

JAR or Module	Project
☑ DonateEJB.jar	DonateEJB
☑ serviceLocatorMgr.jar	

3. Save and close.

7.4 Test the Donate session EJB with the UTC

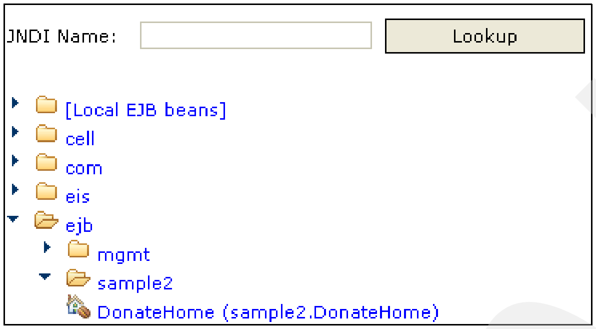
In this section you use the UTC to test the Donate session EJB using the remote interface.

1. Restart the **UTC** using the same steps as described in section “6.7.1 Restart the UTC” on page 158.

> **Note:** Remember to set the user and password fields under **WAS v6 ClassLoading**.

2. From the **Test Client** in the upper left, select **JNDI Explorer**.

3. From the **JNDI Explorer** pane (in the right) locate the reference to the entity EJB remote home interface by expanding to **ejb** → **sample2**, and left-clicking **DonateHome**.

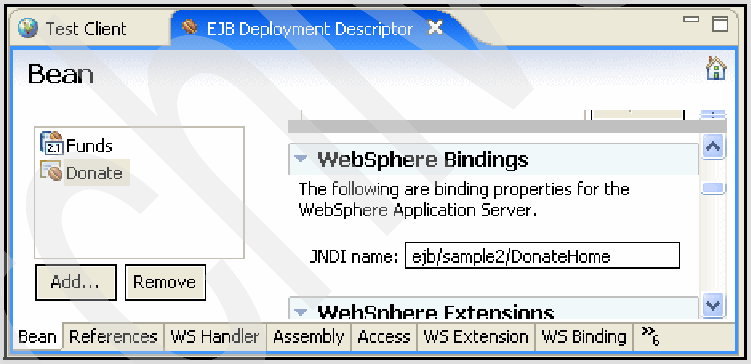


JNDI Name: [ ] Lookup

- [Local EJB beans]
- cell
- com
- eis
- ejb
  - mgmt
  - sample2
    - DonateHome (sample2.DonateHome)

Behind the scenes:

You are accessing the EJB through the remote interface, so you use the direct JNDI name (ejb/sample2/DonateHome). You can view this JNDI name from the bean tab on the Donate2 EJB deployment descriptor.

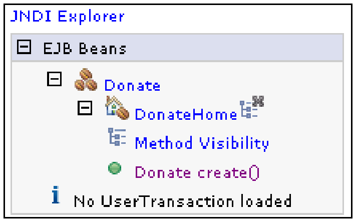


The following are binding properties for the WebSphere Application Server.

JNDI name: ejb/sample2/DonateHome

The Employee and Funds entity EJBs were implemented with the local interface; therefore, they do not have a JNDI name and can only be located by a local EJB name or reference.

4. From the **EJB Beans** section (on the left), expand to **EJB Beans** → **Donate** → **DonateHome**, and click on **Donate create()**.



JNDI Explorer

⊟ EJB Beans

⊟ Donate

⊟ DonateHome

Method Visibility

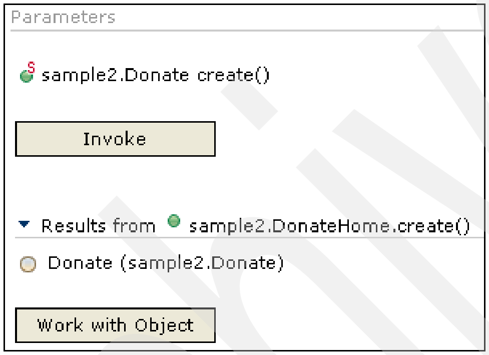
Donate create()

No UserTransaction loaded

5. From the resulting right pane, perform the following:

   – Click **Invoke**.

   – Click **Work with object**.



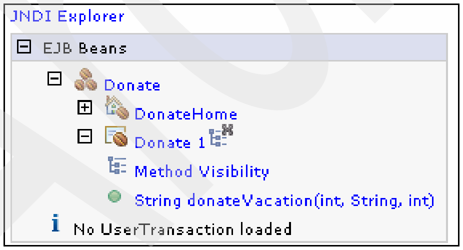
Parameters

sample2.Donate create()

Results from sample2.DonateHome.create()

Donate (sample2.Donate)

6. Expand on the left to **EJB Beans** → **Donate** → **Donate1**, and click **String donateVacation (xxx)**.



JNDI Explorer

⊟ EJB Beans

⊟ Donate

⊞ DonateHome

⊟ Donate 1

Method Visibility

String donateVacation(int, String, int)

No UserTransaction loaded

7. From the resulting right pane perform the following:

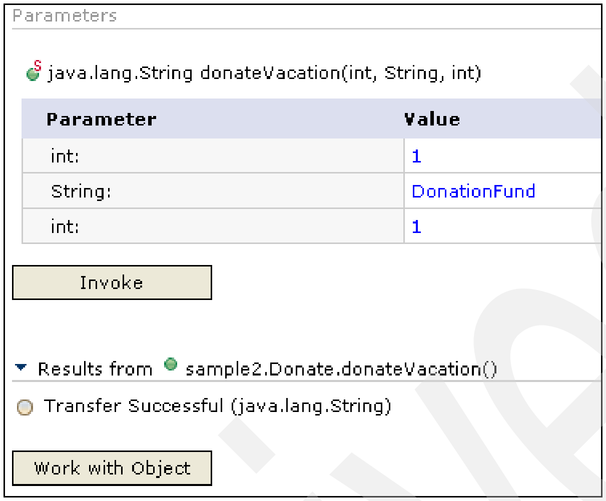
   – Under the **Value** column in the first row (for the employee ID) enter **1**.

   – Under the **Value** column in the second row (for the fund name) enter **DonationFund**.

   – Under the **Value** column in the third row (for the transfer amount) enter **1**.

– Click **Invoke**.

The resulting message should be “Transfer Successful”. This text came from the donateVacation method that you created in DonateBean.java.


Parameters

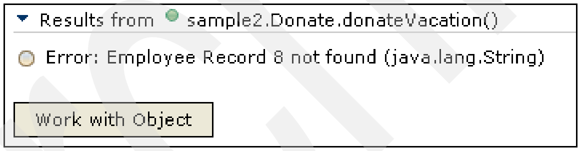
java.lang.String donateVacation(int, String, int)

Parameter	Value
int:	1
String:	DonationFund
int:	1

Results from sample2.Donate.donateVacation()

Transfer Successful (java.lang.String)

8. 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.


Results from sample2.Donate.donateVacation()

Error: Employee Record 8 not found (java.lang.String)

Behind the scenes:

Notice that the **Console** view displays a Java stack trace when an error with finding the Employee or Funds object occurs:

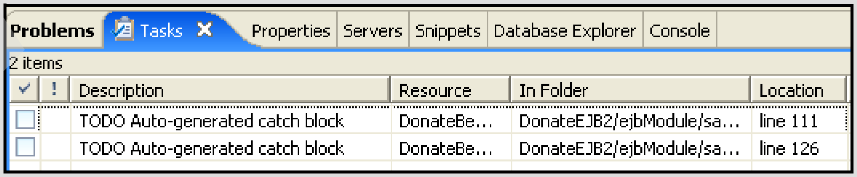
```
[7/5/06 10:59:55:862 CDT] 0000002b SystemErr     R
javax.ejb.ObjectNotFoundException: sample.EmployeeKey@8
[7/5/06 10:59:55:955 CDT] 0000002b SystemErr     R at
com.ibm.ejs.container.EJSHome.activateBean_Common(EJSHome.java:
1923)
[7/5/06 10:59:55:955 CDT] 0000002b SystemErr     R at
com.ibm.ejs.container.EJSHome.activateBean_Local(EJSHome.java:1
782)
[7/5/06 10:59:55:955 CDT] 0000002b SystemErr     R at
sample.EJSCMPEmployeeHomeBean_8b0a9aca.findByPrimaryKey_Local(E
JSCMPEmployeeHomeBean_8b0a9aca.java:91)
```

This occurs because the EJB find snippet code prints out a stack trace when it detects a FinderException when looking up the entity EJB:

```
EmployeeLocalHome anEmployeeLocalHome = (EmployeeLocalHome)
      ServiceLocatorManager.getLocalHome(
         STATIC_EmployeeLocalHome_REF_NAME,
         STATIC_EmployeeLocalHome_CLASS);
try {
   if (anEmployeeLocalHome != null)
      return anEmployeeLocalHome.findByPrimaryKey(primaryKey);
} catch (javax.ejb.FinderException fe) {
   // TODO Auto-generated catch block
   fe.printStackTrace();
}
return null;
```

Note the // TODO statement in the catch block. This is how AST V6.1 indicates that the following code line(s) is temporary and, for real use, should be replaced by the developer with the proper error handling.

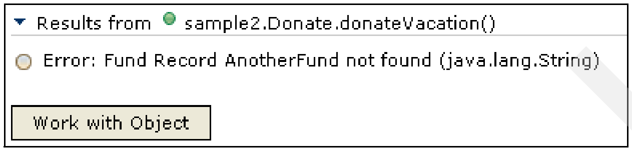
AST V6.1 lists all the //TODO statements in the **Task** view in the lower-right pane.



2 items

✓	!	Description	Resource	In Folder	Location
□		TODO Auto-generated catch block	DonateBe...	DonateEJB2/ejbModule/sa...	line 111
□		TODO Auto-generated catch block	DonateBe...	DonateEJB2/ejbModule/sa...	line 126

9. 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.



```
▼ Results from ● sample2.Donate.donateVacation()

○ Error: Fund Record AnotherFund not found (java.lang.String)
```

10. Close the UTC.

Extra credit:

Use the following steps to verify that the Funds entity EJB was updated:

1. From the **Test Client** in the upper left, select **JNDI Explorer**.
2. From the **JNDI Explorer** pane (in the right), locate the reference to the entity EJB local home interface by expanding to **[Local EJB Beans]** → **ejb** → **ejbs**.
3. Left-click **FundsLocalHome**.
4. From the **EJB Beans** section (on the left), locate the desired entity EJB instance by expanding to **EJB Beans** → **FundsLocal** → **FundsLocalHome** and clicking on **findbyPrimaryKey(FundsKey)**.
5. From the resulting right pane perform the following:
   - Change the right most selection from **Constructors** to **FundsKey(String)**.
   - Expand the left most selection **to FundsKey** → **Constructor:FundsKey** → **String**.
   - Enter “**DonationFund**” under the **Value** column, and click **Invoke**.
   - Work with the returned object (for primary key = 1) by clicking on **Work with object**.
6. Work with the entity EJB instance by expanding on the left to **EJB Beans** → **FundsLocal(ejbs.FundsKey@xxxxxxxx)**.
   - Select the Integer **getFundbalance()** method from the left pane, and from the right pane perform the following:
     - Click **Invoke**.
     - View the result (in the bottom half of the right pane).

The current vacation value should be 1001 (the initial value of 1000 set in section 6.7.2, “Test the Funds entity EJB with the UTC” on page 158 and the 1 day transferred above).

7.5 Explore!

Explore resources are available at the following Web site:

- IBM developerWorks: EJB Programming with snippets in WebSphere Studio V5.1.2

http://www.ibm.com/developerworks/websphere/library/techarticles/0410_schacher/0410_schacher.html

Note: Although this link is dated (October, 2004), it still accurately describes the differences between JNDI lookups using the J2EE standard InitialContext.lookup method and the IBM specific ServiceLocatorManager.getxxx methods.

8

Create the employee facade business logic element

In this chapter you implement usage patterns (session facades and data objects) that are a source of debates in the J2EE community. For the purpose of this book, together they provide capabilities that can be used in Chapter 11, “Implement core security” on page 257 to allow mediated access to back-end data.

8.1 Learn!

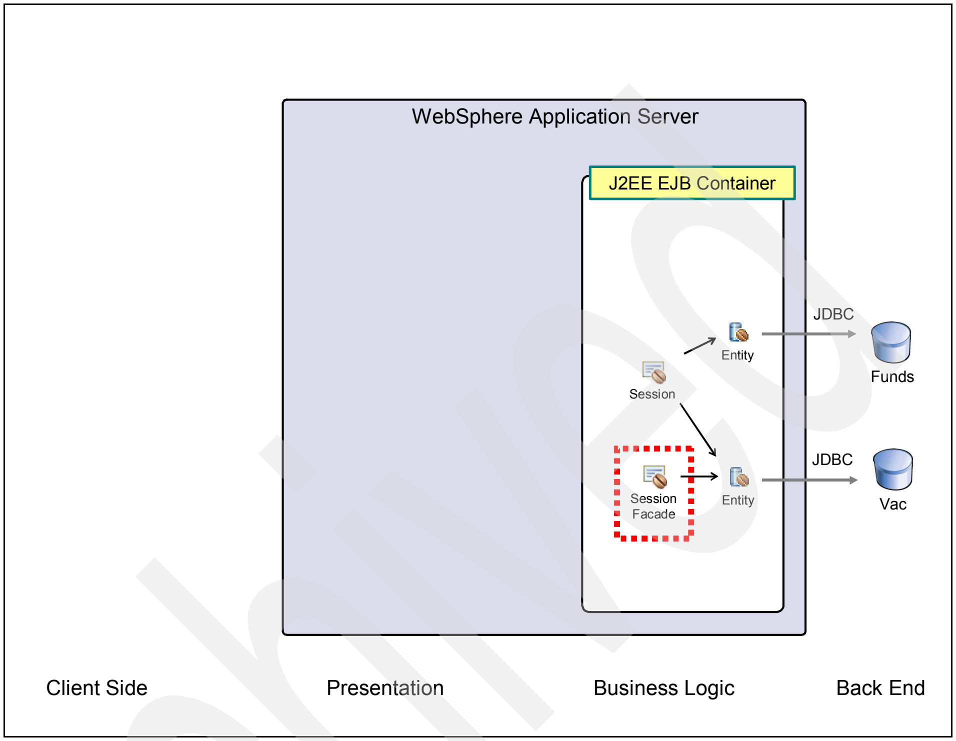


Figure 8-1 Donate application: EmployeeFacade session EJB

Note: This redbook is not using the SessionFacade EJB wizard shipped with Rational Application Developer that creates a session facade EJB using the Service Data Object (SDO) because that wizard is not shipped with AST V6.1.

AST V6.1 also lacks any SDO tooling. Therefore, this book uses the standard session EJB tooling to create a session EJB method that returns a serializable Java class.

Reference section 8.5, “Explore!” on page 209 for further information.

Session Facades provide optional mediated access between the user application and the actual back-end data element:

- The client initializes to the Session Facade, typically using the remote interface.

- The Session Facade method initializes to the entity EJB (typically using the local interface) and then invokes the find method to retrieve the record.
- The Session Facade method creates an instance of a serializable Java class that represents the fields in the entity EJB and copies the values from the entity EJB into the serializable Java class.
- The Session Facade returns the serializable Java class to the client, which reads the fields using normal local Java calls.

For example, in Chapter 11, "Implement core security" on page 257, you will add code to check the user role and only return salary information to the users in the DMANAGERS role. This type of mediated access is not possible in the entity EJB.

Session Facades are also useful because many higher-level tooling packages such as Web services (contained here in AST V6.1) and JavaServer Faces (contained in Rational Application Developer V6.x) only support session EJBs and do not support the direct access of entity EJBs.

Implementation of the Session Façade does not prevent direct access of the entity EJB by other components. Typically the entity EJB is exposed only through the EJB local interface. This means that it can only be accessed by other artifacts in the same J2EE application server instance, thus preventing direct access from other programs. Meanwhile the Session Façade is exposed through the EJB remote interface, which means it can be accessed by any program.

Data objects (in this context a serializable Java class) combine the individual fields in the entity EJB into a single, data object. This allows the application program to request and manipulate the entity EJB in the context of a single object instead of on a field-by-field basis.

- Without data objects, the application must get (or set) on a field-by-field basis. If an entity EJB has 10 fields, the application must issue 10 *gets* to read the entire entity EJB record.
- With data objects, the application can *get* (or set) the entire Data record with a single read.

Additional learn resources are available at the following Web site:

- IBM developerWorks: The top 10 (more or less) J2EE best practices: Always use Session Facades whenever you use EJB components:

  http://www.ibm.com/developerworks/websphere/techjournal/0405_brown/0405_brown.html#sec6

8.2 Define the EmployeeFacade session EJB

The session facade EJB needs to be defined in a EJB project, and it is appropriate to put it in the same project as the entity EJB.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB** → **DonateEJB** → **Session Beans**, and right-click **New** → **Session Bean**.
2. At the **Create an Enterprise Bean** pop-up, perform the following actions:
   - Select **Session bean**
   - **EJB Project**: **DonateEJB**
   - **Bean Name**: **EmployeeFacade**
   - **Source folder**: **ejbModule**
   - **Default package**: **facade**
   - IMPORTANT! Ensure that **Generate an annotated bean class** is cleared.
   - Click **Next**.
3. At the **Enterprise Bean Details** pop-up, review the default values that define a **Remote client view** only—you should not have to make any changes on this pop-up.
4. Click **Finish**.

8.2.1 Create the EJB reference

The getEmployeeByKey method that you will create in the next section accesses the Employee entity EJB. Per J2EE best practices, you access this by using an EJB reference instead of a direct JNDI lookup. The following steps are similar to those used in section 7.2.2, “Create the EJB references” on page 172.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB** → **DonateEJB,** and double-click to open the deployment descriptor.
2. From the resulting **EJB Deployment Descriptor** editor, switch to the **References** tab.
3. Add the Employee EJB reference by performing the following tasks:
   - Under References, select **EmployeeFacade**, and click **Add...**
   - At the resulting **Reference** pop-up, select **EJB reference**, and click **Next**.
   - At the **Add EJB Reference** pop-up perform the following:
     - Select **Enterprise Beans in the workspace**.
     - Select **DonateEAR** → **DonateEJB** → **Employee**. Note that the **Name** field is automatically populated with **ejb/Employee**.

- Set **Ref Type** to **Local.**

4. Click **Finish**.
5. Save and close the **EJB deployment descriptor**.

8.3 Code the EmployeeFacade session EJB

In this section you add your business logic to the generated session facade EJB, which at this point contains only the infrastructure code needed to initialize the overall bean.

8.3.1 Create the Employee JavaBean

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB** → **ejbModule** → **facade**, and right-click **New** → **Class**.
2. At the **Java Class** pop-up, set **Name** to **Employee**, and click **Finish**.
3. At the resulting **Employee.java editor** perform the following:

   – Modify the class definition to include implements Serializable (the change is in bold below):

   ```
   public class Employee implements Serializable {
   }
   ```

   – Insert the required variable definitions (the changes in bold below):

   ```
   public class Employee implements Serializable {
      private int employeeid;
      private String firstname;
      private String middlename;
      private String lastname;
      private int vacamount;
      private BigDecimal salary;

   }
   ```

   – Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**):

   - At the **Organize Imports** pop-up, select **java.io.Serializable**, and click **Next**.
   - Still on the **Organize Imports** pop-up, select **java.math.BigDecimal**, and click **Finish**.

Behind the scenes:

It is critical to select the proper classes when organizing imports. Errors may not be evident until another method/class references the artifact that is qualified by this import.

For example, in the above example, no errors would be visible at this point if you selected com.ibm.icu.math.BigDecimal instead of java.math.BigDecimal. Recall that this is used for the salary field (and the associated getter and setter methods).

However, when you import the following line

```
emp0.setSalary(anEmployeeLocal.getSalary());
```

with FRAG2 in section 8.3.2, “Create the getEmployeeByKey method” on page 201, the following error is displayed:

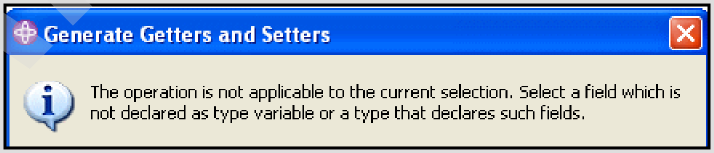
```
The method setSalary(BigDecimal) in the type Employee is not
applicable for the arguments (BigDecimal)
```

This indicates that the type required by the setSalary method in the Employee object (com.ibm.icu.math.BigDecimal) is different than the type returned by the getSalary method in the Employee entity EJB (java.math.BigDecimal).

– On the editor, right-click anywhere with the class body (for example, from public class Employee... down to the closing bracket), and select **Source → Generate Getters and Setters...**

Off course?

If you receive the following pop-up, you selected outside the class body. Try clicking somewhere else in the editor, such as right on the public class Employee... line.

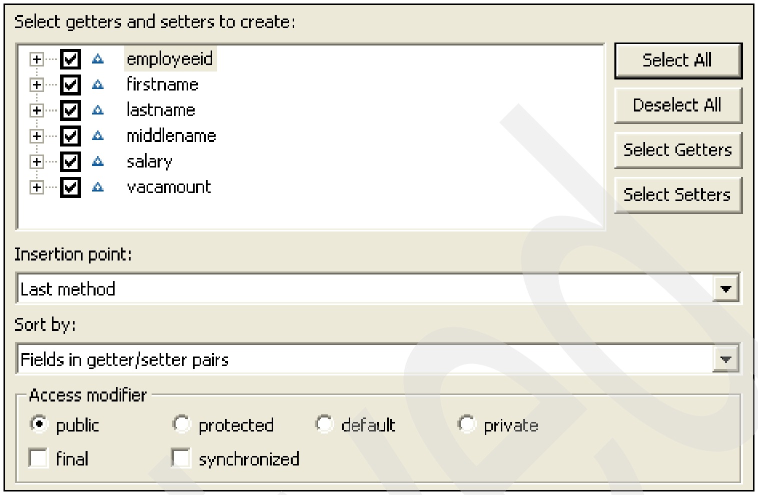


The operation is not applicable to the current selection. Select a field which is not declared as type variable or a type that declares such fields.

– At the **Select getters and setters to create** pop-up, perform the following:

- Select **Select All**, which selects all the variables.

- Ensure that **Access modifier** is set to **public**.
- Click **OK**.



Select getters and setters to create:

- ☑ employeeid
- ☑ firstname
- ☑ lastname
- ☑ middlename
- ☑ salary
- ☑ vacamount

Insertion point: Last method

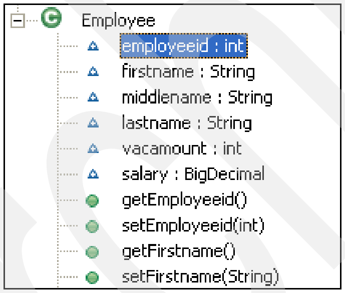
Sort by: Fields in getter/setter pairs

Access modifier

● public ○ protected ○ default ○ private

□ final □ synchronized

– Back at the **Employee.java** editor, note the new get* and set* methods that were added to the class. These are also visible in the **Outline** view.



```
Employee
    employeeid : int
    firstname : String
    middlename : String
    lastname : String
    vacamount : int
    salary : BigDecimal
    getEmployeeid()
    setEmployeeid(int)
    getFirstname()
    setFirstname(String)
```

4. Save and close **Employee.java**.

8.3.2 Create the getEmployeeByKey method

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB** → **DonateEJB** → **Session Beans** → **EmployeeFacade** → **EmployeeFacadeBean**(.java), and double-click to open the Java class.
2. In the **EmployeeFacadeBean.java** editor, insert a blank line before the closing bracket.
3. Position the cursor on this new line, and add the following method definition:

```
public Employee getEmployeeByKey(EmployeeKey primaryKey)
{
   //* Insert call to Employee entity EJB.

   //* Set the Employee java bean

}
```

4. Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux) and organize the imports (**Ctrl+Shift+O**) to remove the error regarding EmployeeKey.

   The error stating “This method must return a result of type Employee” remains. You will fix this in a subsequent step.

5. Add the call to the Employee entity EJB using a snippet:
   - Select the line after **//* Insert call to Employee entity EJB**.
   - In the **Snippets** view (lower right), expand the **EJB** category, and double-click **Call an EJB “find” method**.
   - At the **Select Reference** pop-up, select **ejb/Employee**, and click **Next**.
   - At the **Enter Parameter Values** pop-up, note that the **primaryKey** value matches the name of the EmployeeKey object used in the getEmployeeByKey, and click **Finish**.

6. Add code to populate the Employee object using snippets, by performing the following tasks:
   - Select the line after **//* Set the Employee java bean**.
   - In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click on **FRAG2: EmployeeFacade logic**. This inserts the snippet code into **EmployeeFacadeBean.java** at the current cursor location.

– Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux).

```
if (anEmployeeLocal != null)
{
   Employee emp0 = new Employee();
   emp0.setEmployeeid(anEmployeeLocal.getEmployeeid());
   emp0.setFirstname(anEmployeeLocal.getFirstname());
   emp0.setMiddlename(anEmployeeLocal.getMiddlename());
   emp0.setLastname(anEmployeeLocal.getLastname());
   emp0.setSalary(anEmployeeLocal.getSalary());
   emp0.setVacamount(anEmployeeLocal.getVacamount());
   return emp0;
}
else
   return null;
```

Figure 8-2 FRAG2: EmployeeFacade logic

All errors should be resolved.

Behind the scenes:

The completed getEmployeeByKey method does the following:

- Locates the Employee entity EJB:

```
//* Insert call to Employee entity EJB.
   EmployeeLocal anEmployeeLocal =
      find_EmployeeLocalHome_findByPrimaryKey(primaryKey);
```

- If the Employee record exists, copies the fields from the entity EJB to the Employee object, and then returns this to the caller:

```
//* Set the Employee java bean
if (anEmployeeLocal != null)
{
   Employee emp0 = new Employee();
   emp0.setEmployeeid(anEmployeeLocal.getEmployeeid());
   emp0.setFirstname(anEmployeeLocal.getFirstname());
   emp0.setMiddlename(anEmployeeLocal.getMiddlename());
   emp0.setLastname(anEmployeeLocal.getLastname());
   emp0.setSalary(anEmployeeLocal.getSalary());
   emp0.setVacamount(anEmployeeLocal.getVacamount());
   return emp0;
}
```

- If the Employee record does not exist, returns null to the caller:

```
else
   return null;
```

7. Switch to the **Outline** view in the upper-right pane, select the **getEmployeeByKey** method, and right-click **Enterprise Bean** → **Promote to Remote Interface**. The letter ‘R’ is added next to this method, indicating that it is available through the remote interface.

Work around:

The **Outline** view may not update with the letter ‘R’ after promoting the getEmployeeByKey method to the remote interface. If this happens, close the **EmployeeFacadeBean.java** editor and reopen. The **Outline** view should be reloaded correctly.

This is a known problem that is fixed in Web Tools Platform 1.5 (and will be included in the AST V6.1 update/version that includes WTP 1.5).

8. Save and close **EmployeeFacadeBean.java**.

   The **Problems** view initially shows several messages, indicating that the EJB deployment code is out of sync with the updated session EJB. The automatic publishing process should regenerate the EJB deployment code in the background and these errors should disappear.

> **Off course**?
>
> If the errors do not completely disappear within approximately a minute, the server may not have detected the latest changes.
>
> You should be able to resolve this by forcing a re-publication:
>
> - Switch to the **Servers** view, select **ExperienceJ2EE Server @ localhost**, and right-click **Publish**.
>
> The publishing process is re-run, and the errors should then disappear.

> **Extra credit**:
>
> The getEmployeeByKey method is read-only: The back-end data is never updated. Instead, the information is copied from the returned entity EJB instance into the Employee object, and the method completes. The caller receives the Employee object, but changes written to that object are not written to the back-end database.
>
> If you wanted to use a session facade to update the back-end database, add another method, such as the setEmployee method, as follows:

```
public void setEmployee(Employee emp0) throws Exception{
   EmployeeKey primaryKey = new EmployeeKey(emp0.getEmployeeid());
   EmployeeLocal anEmployeeLocal =
         find_EmployeeLocalHome_findByPrimaryKey(primaryKey);
   // * Set the Employee entity EJB
   if (anEmployeeLocal != null) {
      anEmployeeLocal.setFirstname(emp0.getFirstname());
      anEmployeeLocal.setMiddlename(emp0.getMiddlename());
      anEmployeeLocal.setLastname(emp0.getLastname());
      anEmployeeLocal.setSalary(emp0.getSalary());
      anEmployeeLocal.setVacamount(emp0.getVacamount());
   } else
      throw new Exception("employee ID " + emp0.getEmployeeid()
         + " not found");
}
```

8.3.3 Work around: Add serviceLocatorMgr.jar to MANIFEST.MF

The jar file contained in DonateEAR contains classes that are used by the EJB “Find” method code added by the snippet. In Rational Application Developer V6.0, this was automatically added to both the classpath and the MANIFEST.MF of the EJB project when running the snippet. In AST V6.1 it is only added to the classpath (and not to the MANIFEST.MF). This is a known problem that is fixed in Web Tools Platform 1.5, and will be included in the AST V6.1 update/version that includes WTP 1.5.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB** → **ejbModule** → **META-INF** → **MANIFEST.MF**, and double-click to open.
2. From the resulting **Jar Dependencies** editor, scroll down to the **Dependencies** section, and select **serviceLocatorMgr.jar**.
3. Save and close.

Work around:

In certain situations the MANIFEST.MF may not exist in DonateEJB. If this happens you can recreate the file using the following instructions:

1. Select **EJB Projects** → **DonateEJB** → **ejbModule** → **META-INF**, and right-click **New** → **File**.
2. From the resulting **New File** pop-up, set **File name** to **MANIFEST.MF** (all upper case), and click **Finish**.
3. From the resulting **Jar Dependencies** editor, scroll down to the **Dependencies** section, and select **serviceLocatorMgr.jar**.
4. Save and close.

8.4 Test the EmployeeFacade session EJB with the UTC

In this section you use the UTC to test the EmployeeFacade session EJB using the remote interface.

1. Restart the **UTC** using the same steps as described in section “6.7.1 Restart the UTC”.

   Remember to set the user and password fields under **WAS v6 ClassLoading**.
2. From the **Test Client** in the upper left, select **JNDI Explorer**.

3. From the **JNDI Explorer** pane (in the right), locate the reference to the entity EJB remote home interface by expanding to **ejb** → **facade**, and left-clicking on **EmployeeFacadeHome**.

4. From the **EJB Beans** section (on the left), expand to **EJB Beans** → **EmployeeFacade** → **EmployeeFacadeHome**, and click **EmployeeFacade create()**.

5. From the resulting right pane, perform the following:

   – Click **Invoke**.

   – Click **Work with object**.

6. Expand on the left to **EJB Beans** → **EmployeeFacade** → **EmployeeFacade 1**, and click **Employee getEmployeeByKey(EmployeeKey)**.

7. From the resulting right pane perform the following:

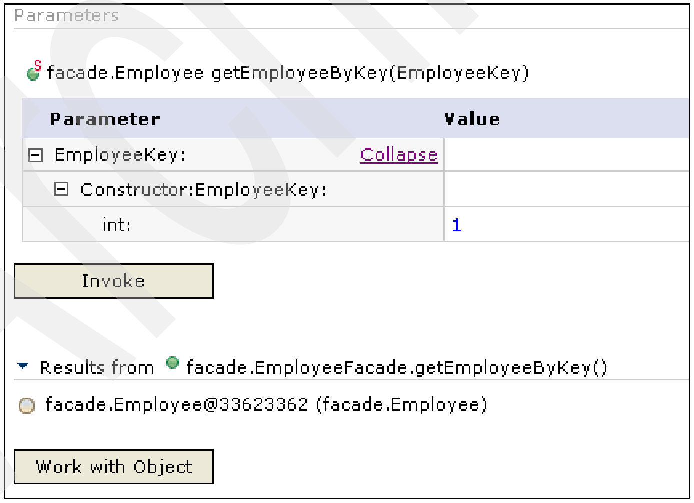
   – Change the right most selection from **Constructors** to **EmployeeKey(int)**.

   – Expand the left most selection **to EmployeeKey** → **Constructor:EmployeeKey** → **int**.

   – Enter "1" under the **Value** column, and click **Invoke**.

   The **Results** section should show the return of an object (facade.Employee@xxxxxxxx (facade.Employee).

   – Click **Work with object**.



Parameters

facade.Employee getEmployeeByKey(EmployeeKey)

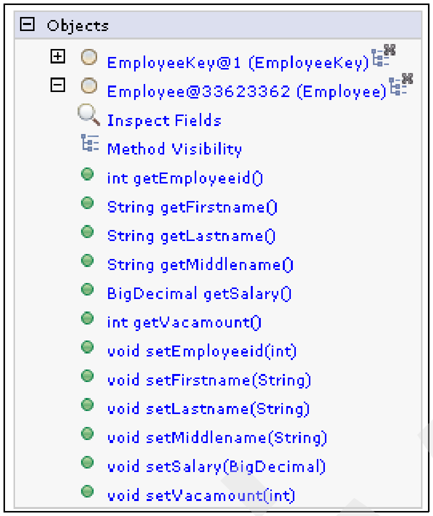
Parameter	Value
⊟ EmployeeKey: Collapse	
⊟ Constructor:EmployeeKey:	
int:	1

Results from facade.EmployeeFacade.getEmployeeByKey()

facade.Employee@33623362 (facade.Employee)

8. Expand on the left **Objects** → **Employee@xxxxxxxx(Employee)**. Note the various field for getting and setting the Employee information. These methods

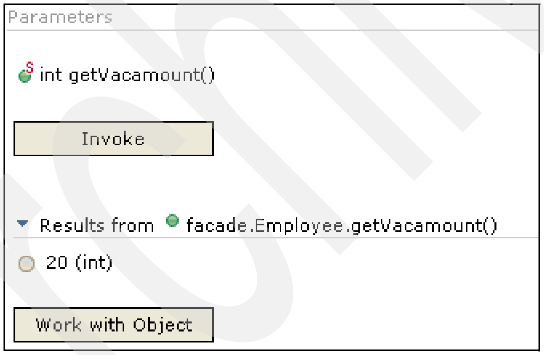
work against a local data object, which means that they can be read and set within the local program— without going back to the EJB container.



Objects

- EmployeeKey@1 (EmployeeKey)
- Employee@33623362 (Employee)
  - Inspect Fields
  - Method Visibility
  - int getEmployeeid()
  - String getFirstname()
  - String getLastname()
  - String getMiddlename()
  - BigDecimal getSalary()
  - int getVacamount()
  - void setEmployeeid(int)
  - void setFirstname(String)
  - void setLastname(String)
  - void setMiddlename(String)
  - void setSalary(BigDecimal)
  - void setVacamount(int)

– Execute a getter method. On the left, click the method **int getVacamount()**, and on the right click **Invoke**.

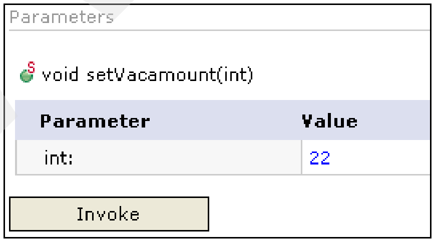


int getVacamount()

Results from facade.Employee.getVacamount()

20 (int)

– Execute a setter method. On the left, click the method **void setVacamount(int)**. On the right enter a **Value** of **22**, and click **Invoke**.



void setVacamount(int)

Parameter	Value
int:	22

9. Close the UTC.

Extra credit:

Using the following instructions, verify that the Employee entity EJB was not changed:

1. From the **Test Client** in the upper left, select **JNDI Explorer**.
2. From the **JNDI Explorer** pane (in the right) locate the reference to the entity EJB local home interface by expanding to **[Local EJB Beans]** → **ejb** → **sample**, and left-clicking on **EmployeeLocalHome**.
3. From the **EJB Beans** section (on the left), locate the desired entity EJB instance by expanding to **EJB Beans** → **EmployeeLocal** → **EmployeeLocalHome**, and clicking on **findbyPrimaryKey(EmployeeKey)**.
4. From the resulting right pane, perform the following:
   - Change the right most selection from **Constructors** to **EmployeeKey(int)**.
   - Expand the left most selection **to EmployeeKey** → **Constructor:EmployeeKey** → **int**. Enter “1” under the **Value** column, and click **Invoke**.
   - Work with the returned object (for primary key = 1) by clicking **Work with object**.
5. Work with the entity EJB instance by expanding on the left to **EJB Beans** → **EmployeeLocal(sample.EmployeeKey@1)**, and selecting the **int getVacamount getter method** from the left-hand pane.
6. From the right-hand pane, click **Invoke**.

   View the result in the bottom half of the right-hand pane.

The current vacation value should be the same as that returned when you executed the **int getVacamount()** method on the EmployeeFacade session EJB (on page 208).

8.5 Explore!

The EmployeeFacade session EJB was implemented using a standard java class (Employee) for the returned data. A certain amount of “manual” coding is required, but this approach is portable to all J2EE application servers.

IBM WebSphere Application Server V6.0 and Rational Application Developer V6.0 support an alternate session facade approach that utilizes the Service Data Object (SDO) technology, which provides a common data access and representation that can be used both across a variety of platforms and languages.

Service Data Objects (SDO):

SDO is similar to XML in that it provides a common format that can be read across a variety of systems and languages (currently native Java and C++). SDO is different than XML in that it represents the information in a binary format instead of text, resulting in improved performance for larger data objects.

The intent is for programmers to access a variety of data sources through the SDO representation: XML, EJBs, Web services, JCA adapters, and so forth. Thus, the programmer implements one set of data access routines that work across all data sources. These data access routines are thus independent of the specific data source used.

The SDO specification is jointly sponsored by both IBM and BEA. A preliminary specification is available on the Java Community Process home page, and was sponsored as an open source effort called Tuscany.

IBM and others have also generated a companion specification called Service Component Architecture (SCA) that focuses on providing a common invocation framework that utilizes SDO.

These two technologies together represent the foundation for a Service Oriented Architecture (SOA) framework. Reference section 18.8, “Service Oriented Architecture” on page 427 for further detail.

Rational Application Developer contains a Create Session Bean Facade wizard that creates a session EJB that uses SDO to represent the data, and provides various methods including getxxxByKey, createxxx, updatexxx, deletexxx, and getAllxxObjects.

Additional explore resources are available at the following Web sites:

- IBM developerWorks: Coarse-grained service pattern with SDOs and EJB Session Facades in Rational Application Developer (Part 1) and (Part 2):

  http://www.ibm.com/developerworks/rational/library/04/r-3311/index.html
  http://www.ibm.com/developerworks/rational/library/05/1122_makin/index.html

- IBM developerWorks: Service Data Objects:

  http://www.ibm.com/developerworks/webservices/library/specification/ws-sdo

- IBM Education Assistant: Service Data Objects (SDO):

  http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.rad_v6/rad/6.0/SDO.html

- Apache Tuscany (open source SCA/SDO project):

  http://incubator.apache.org/tuscany

Create the Web front end

In this chapter you create the JSPs that provide the Web interface for interacting with the EmployeeFacade and Donate session EJBs created in the previous chapters.

9.1 Learn!

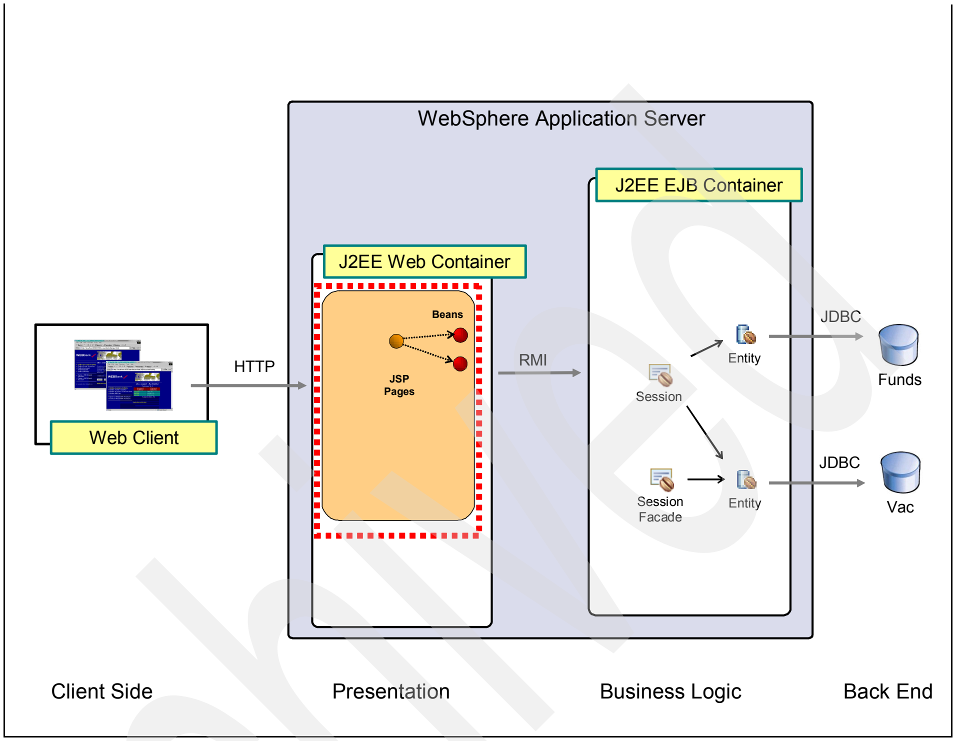


Figure 9-1 Donate application: Web front end

In general, an overall *Web front end* in J2EE usually consists of the following:

- Web components. These are the actual artifacts that are invoked when a user types a Web address into a browser. In a J2EE environment the addressable Web components can include the following:
  - Static content such as HTML pages or image files. Note that these are not included in the J2EE 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 J2EE artifacts
  - Servlets that provide 100% dynamic content. A servlet is a specific java class that implements javax.servlet.Servlet. Following is an extract of a servlet method for an HTTP post operation:

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
   PrintWriter out = response.getWriter();
   out.println("<body>");
   out.println("<h1>Hello World2</h1>");
   out.println("</body>");
}
```

  The servlet is mapped to a URI through definitions in the Web deployment descriptor.

  The one weakness of servlets is that all output occurs using out statements (rather than through HTML tags). Thus, typically you cannot use WYSIWYG tools to visually create and compose a servlet.

- – JavaServer Pages (JSPs), which provide for “partial dynamic” content. The JSP consists of both HTML statements and java statements:

```
<jsp:useBean id="record" class="donate.web.DonateWebRecord"
scope="session"/>
<jsp:setProperty name="record" property="donation_Amount"/>
<h1>Donation Result</h1>
<br>
<%=record.processDonation()%>
```

  At runtime this file is complied into a servlet and invoked.

  The JSP is placed in the WebContent directory (in the Web module) and is implicitly mapped to the context root of the Web module.

  JSPs can be developed using WYSIWYG tools.

- Session store, which maintains persistence information between the invocation of various Web components:

  – 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.

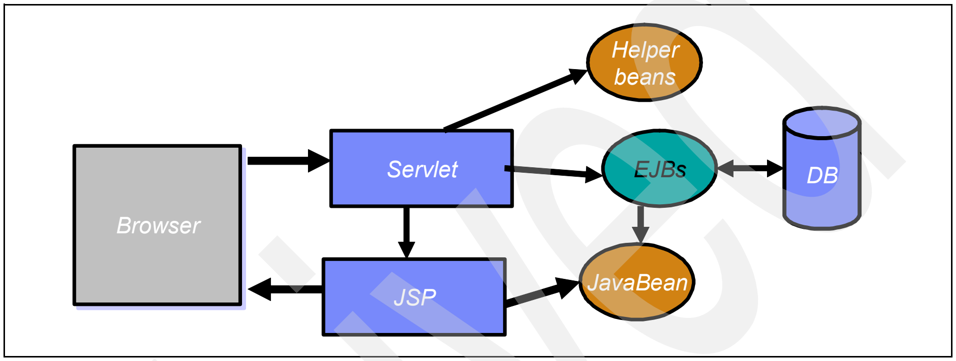
  – On the server side, the session information is maintained in a persistence store (typically in memory, but also potentially in a relational database or in a publish/subscribe mechanism).

  The first artifact invocation initializes the store and returns the session id back to the browser. On subsequent artifact invocations, this session id is used to retrieve any stored information from the session store.

- Controller, which acts as an intermediary between the request and the actual final page.

  In most application scenarios, the actual processing of session information and invocation of “user” logic (for example what back-end EJBs to invoke and how to determine what page to go to next) is somewhat common across the various artifacts. Therefore, rather than embed all this into various JSPs and servlets, this “control” logic is normally put into a single servlet (called the controlling servlet) which processes the request from the browser, invokes any back-end functions, and then redirects to the appropriate output page (which is typically a JSP).

These components are used to implement a Model View Controller (MVC) architecture, which allows for the appropriate usage of J2EE technologies (servlets for programming, JSPs for rendering the page):



The above description of MVC is an architectural approach: J2EE provides the required technical components, but the developer still must manually create quite a bit of code to coordinate the overall application.

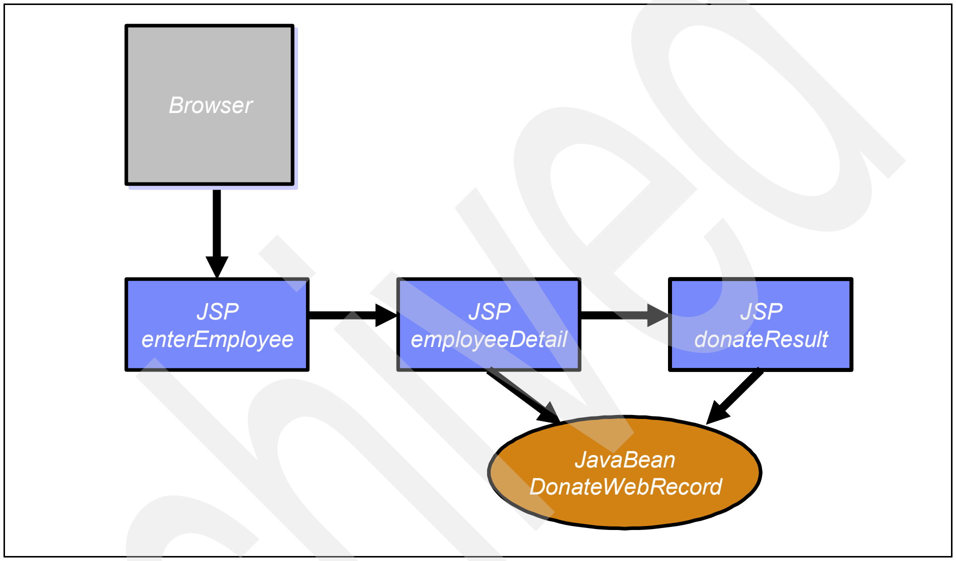
The open source community developed a framework called Struts that defined the artifacts and required runtime support needed for a flexible MVC oriented “Web application”. This framework has been very popular and is shipped with many J2EE runtimes and J2EE IDEs. Note that Struts is not a standard or specification: It is simply an implementation that has been widely adopted. WAS V6.1 provides Struts runtime support and Rational Application Developer V6.0 provides Struts tooling.

The J2EE community recently developed the JavaServer Faces (JSF) specification which provides similar support. JSF is not part of J2EE V1.4 but many J2EE V1.4 runtime and IDE vendors do ship JSF with their products. JSF is part of Java EE 5. WAS V6.1 provides JSF runtime support and Rational Application Developer V6.0 provides JSF tooling.

Reference section 9.5, “Explore!” on page 243 for further information on JSF.

AST V6.1, and the WTP V1.0 release it is based on, has very basic tooling support for Web development: Visual editing of Web deployment descriptors, but no WYSIWYG support for HTML or JSPs, and no Struts/JSF support. Realistically, very few developers will use AST V6.1 for developing complex Web front ends, and instead utilizes IDEs that do provide this type of support, such as Rational Application Developer.

Therefore, this book provides a very rudimentary Web application. This is not a MVC front end, but it does utilize basic JSP capabilities and session support. The browser invokes each page directly without going through a controller, and the page retrieves the DonateWebRecord object from the session bean and utilizes this for both the data and logic.



Clearly, this is NOT a Web front end that demonstrates best programming practices. However, given the level of tooling provided with AST V6.1, this does provide a complete Web front end (for the purpose of the scenario in this book) that provides the basic capabilities you will need to implement J2EE security for the Web components in Chapter 11, “Implement core security” on page 257:

Additional learn resources are available at:

- The J2EE 1.4 Tutorial: Chapter 11: Java Servlet Technology and Chapter 12: JavaServer Pages Technology

  `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

- Wikipedia: Model-view-controller:

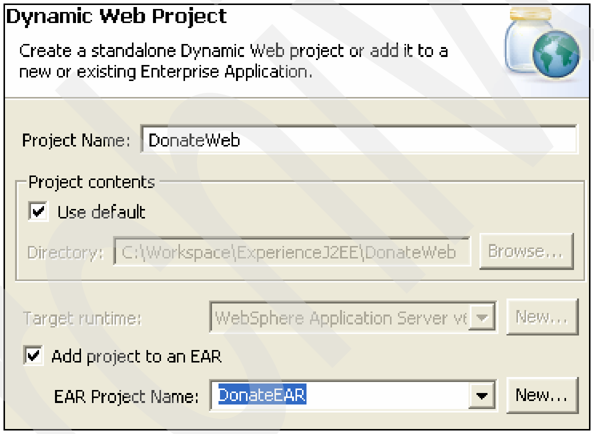
  `http://en.wikipedia.org/wiki/Model-view-controller`

9.2 Create/Configure the Web project

JSPs and servlets that are deployed to a Web container are packaged in a Web project.

9.2.1 Create the Web project

1. From the **Project Explorer**, select **Dynamic Web Projects** and right-click **New** → **Dynamic Web Project**.
2. At the **Dynamic Web Project** pop-up, perform the following:
   - **Name**: **DonateWeb**
   - **Target runtime**: **WebSphere Application Server v6.1**
   - Select **Add Module to an EAR project**.
   - **EAR Project**: **DonateEAR**
   - Click **Finish**.



Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project Name: DonateWeb

Project contents

☑ Use default

Directory: C:\Workspace\ExperienceJ2EE\DonateWeb

Target runtime: WebSphere Application Server v6

☑ Add project to an EAR

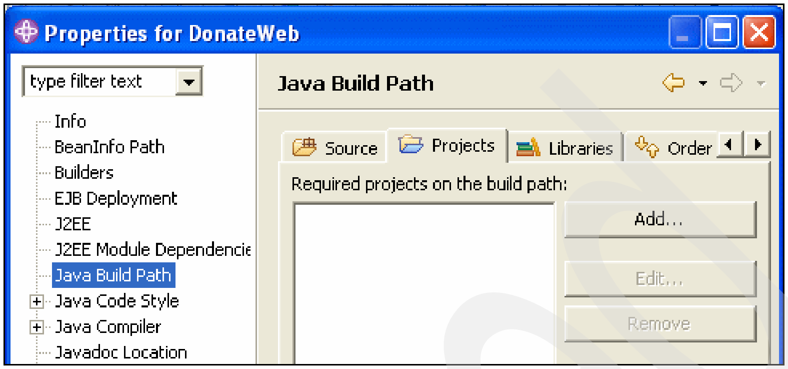
EAR Project Name: DonateEAR

9.2.2 Configure the Web project

You must add DonateEJB to the Java build path for DonateWeb. This is needed for 9.3, "Develop the DonateWebRecord JavaBean" on page 219, which contains a reference to the Employee class you created in DonateEJB (in section 8.3.1, "Create the Employee JavaBean" on page 199).

1. From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** and right-click **Properties**.
2. From the **Properties for DonateWeb** pop-up, perform the following actions:

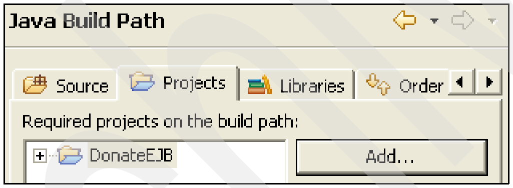
- On the left, click **Java Build Path**.
- On the right, on the resulting **Java Build Path** pane, select the **Projects** tab, and **click Add...**.



Required projects on the build path:

- At the resulting **Select projects to add** pop-up, select **DonateEJB** and click **OK**.

- At the **Properties for DonateWeb** pop-up, note that **DonateEJB** now appears on the **Required projects on the build path** list, and click **OK**.



Required projects on the build path:

9.3 Develop the DonateWebRecord JavaBean

The DonateWebRecord JavaBean is the common object that is shared across all JSPs (or servlets) within the session.

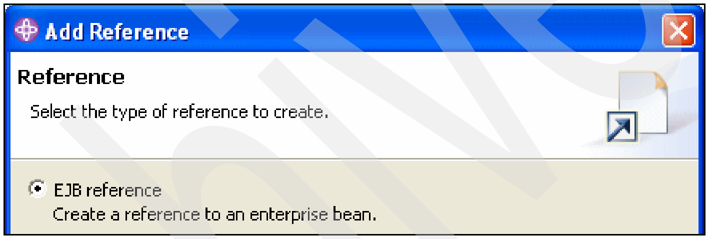
For simplicity, you create a single JavaBean that contains both data, such as the Employee employee_Details variable, and methods, such as processEmployee.

However, best programming practices suggest that you separate data and methods into separate classes.

9.3.1 Create the EJB references

The methods in DonateWebRecord access the EmployeeFacade and Donate session EJBs that you developed in previous chapters. Per J2EE best practices, you access these by using EJB references, instead of a direct JNDI lookup. This is similar to the local EJB references you used in section 7.2.2, “Create the EJB references” on page 172 and in section 8.2.1, “Create the EJB reference” on page 198.

1. From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **DonateWeb**, and double-click to open the deployment descriptor.
2. From the resulting **Web Deployment Descriptor** editor, switch to the **References** tab.
3. Add the EmployeeFacade EJB reference, by performing the following actions:
   - Under References, click **Add...**.
   - At the resulting **Reference** pop-up, select **EJB reference**, and click **Next**.



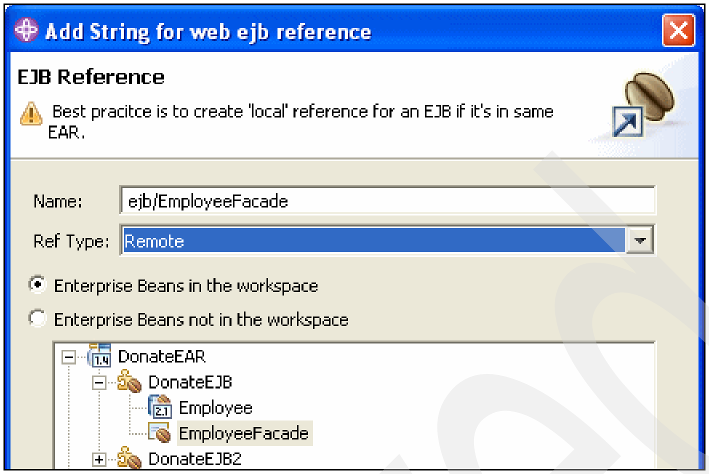
Add Reference

Reference

Select the type of reference to create.

● EJB reference
Create a reference to an enterprise bean.

   - At the **EJB Reference** pop-up, perform the following actions:
     - Select **Enterprise Beans in the workspace**.
     - Expand to/select **DonateEAR** → **DonateEJB** → **EmployeeFacade**. Note that the **Name** field will automatically be filled in with **ejb/EmployeeFacade**.
     - Set **Ref Type** to **Remote**.
   - Click **Finish**.



Add String for web ejb reference

EJB Reference

Best pracitce is to create 'local' reference for an EJB if it's in same EAR.

Name: ejb/EmployeeFacade

Ref Type: Remote

● Enterprise Beans in the workspace

○ Enterprise Beans not in the workspace

Behind the scenes:

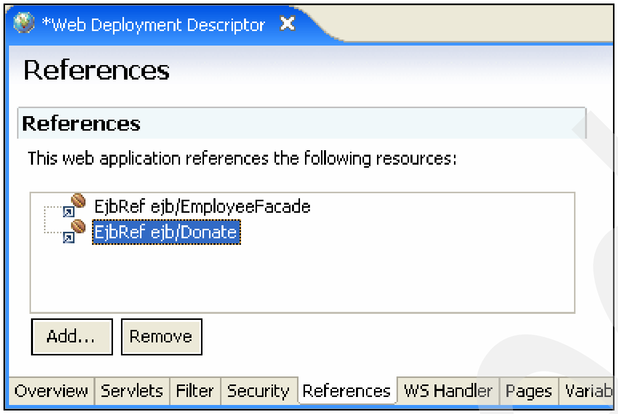
Note the warning **Best practice** [sic] **is to create 'local' reference for an EJB if it's in same EAR**.

Since an EAR file is usually deployed as a single unit, the Web container and the EJB container are running in the same JVM, hence a local interface could be used and would be more efficient.

However, typical deployments in a cluster tend to install Web projects on J2EE server instances in one tier and in EJB projects on J2EE server instances in a second tier. This type of separation requires usage of the remote interface.

4. Add the Donate EJB reference by performing the following tasks:
   - Under References, click **Add...**
   - At the resulting **Reference** pop-up, select **EJB reference**, and click **Next**.
   - At the **EJB Reference** pop-up, perform the following actions:
     - Select **Enterprise Beans in the workspace**.
     - Select **DonateEAR** → **DonateEJB2** → **Donate**. Note that the **Name** field is automatically populated with **ejb/Donate**.
     - Set **Ref Type** to **Remote**.
   - Click **Finish**.

5. When finished, the reference screen should appear similar to the following graphic:



References

References

This web application references the following resources:

- EjbRef ejb/EmployeeFacade
- EjbRef ejb/Donate

6. Save and close the Web deployment descriptor.

Behind the scenes:

The EJB reference definition is stored in two separate files:

- **Web Modules** → **DonateWeb** → **WebContent** → **WEB-INF** → **web.xml** contains the definition of the EJB that is referenced:

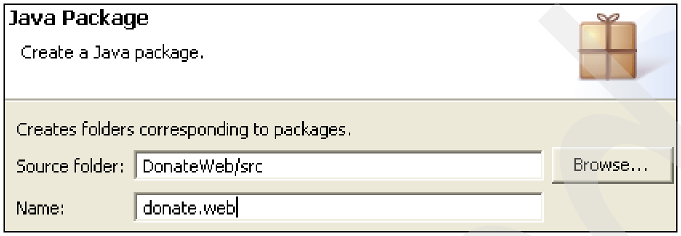
```
<ejb-ref id="EjbRef_1157751535937">
   <description>
   </description>
   <ejb-ref-name>ejb/EmployeeFacade</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>facade.EmployeeFacadeHome</home>
   <remote>facade.EmployeeFacade</remote>
   <ejb-link>DonateEJB.jar#EmployeeFacade</ejb-link>
</ejb-ref>
```

- **Web Modules** → **DonateWeb** → **WebContent** → **WEB-INF** → **ibm-web-bnd.xmi** contains the EJB JNDI name:

```
<ejbRefBindings xmi:id="EjbRefBinding_1157751535937"
         jndiName="ejb/facade/EmployeeFacadeHome">
   <bindingEjbRef href="WEB-INF/web.xml#EjbRef_1157751535937"/>
</ejbRefBindings>
```

9.3.2 Create the base class

1. From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **src**, and right-click **New** → **Package**.
   - At the **Java Package** pop-up, set **Name** to **donate.web**, and click **Finish**.



Java Package

Create a Java package.

Creates folders corresponding to packages.

Source folder: DonateWeb/src

Name: donate.web

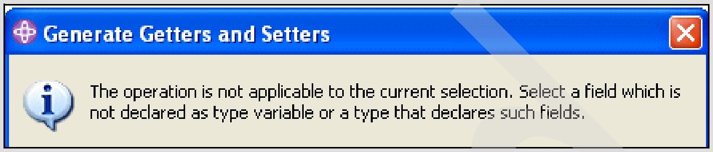
2. Back at the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **src** → **donate.web**, and right-click **New** → **Class**.
   - At the **Java Class** pop-up, set **Name** to **DonateWebRecord**, and click **Finish**.
3. At the resulting **DonateWebRecord.java** editor, perform the following actions:
   - Insert the following highlighted lines:

```
package donate.web;

public class DonateWebRecord {
   private Integer employee_Number;
   private Employee employee_Details;
   private Integer donation_Amount;
}
```

   - Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux) and organize the imports (**Ctrl+Shift+O**) to remove the error for Employee.
   - Right-click anywhere on the editor, and select **Source** → **Generate Getters and Setters...**

Off course?

If you receive the following pop-up, you selected outside the class body. Try clicking somewhere else in the editor, such as right on the private Employee... line.



Generate Getters and Setters

The operation is not applicable to the current selection. Select a field which is not declared as type variable or a type that declares such fields.

- At the **Select getters and setters to create** pop-up, perform the following actions:
  - Select **Select All** to select all the variables.
  - Ensure that **Access modifier** is set to **public**.
  - Click **OK**.
- Return to the **DonateWebRecord.java** editor, and note the new get* and set* methods that were added to the class. These are also visible in the **Outline** view.

9.3.3 Code the processEmployee method

This method is used on the employeeDetail.jsp page to take the input employee_Number, which was entered on the enterEmployee.jsp page, and lookup the employee record.

1. On the **DonateWebRecord.java** editor, insert a blank line before the closing bracket, and position the cursor on this new line.
2. In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG3: DonateWebRecord processEmployee**. This inserts the snippet code into **DonateWebRecord.java** at the current cursor location.

```
public String processEmployee() {
   EmployeeKey primaryKey = new EmployeeKey();
   primaryKey.setEmployeeid(this.getEmployee_Number());

   Employee anEmployee = null;
   String message;

   //* Insert call to Session bean service method

      if (anEmployee == null)
   {
      message = "Error: retrieved null record!";
      anEmployee = new Employee();
   }
   else
   {
      message = "Record retrieved successfully";
   }

   this.setEmployee_Details(anEmployee);
   return message;
}
```

Figure 9-2   FRAG3: DonateWebRecord processEmployee

3. Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**) to remove most of the errors. Some of the errors will remain until you add the call to the EmployeeFacade session EJB.
4. Add the call to the EmployeeFacade session EJB using a snippet. Perform the following tasks:
   – Select the line after **//* Insert call to Session bean service method**.
   – In the **Snippets** view (lower right), expand the **EJB** category, and double-click **Call a Session bean service method**.
   – At the **Select Reference** pop-up, select **ejb/EmployeeFacade**, and click **Next**.
   – At the **SelectMethod Values** pop-up, select **getEmployeeByKey(EmployeeKey primaryKey)**, and click **Finish**.

Behind the scenes:

The **Call a Session bean service method** snippet makes changes similar to those made by the **Call an EJB "find" method** snippet used in section 7.3.2, "Add the call to the Employee entity EJB using a snippet" on page 181.

5. One error remains ("Duplicate local variable of anEmployee"). Fix this by changing this line:

```
Employee anEmployee =
anEmployeeFacade.getEmployeeByKey(primaryKey);
```

   to the following:

```
anEmployee = anEmployeeFacade.getEmployeeByKey(primaryKey);
```

6. Replace the generated catch block, which only catches remote exceptions, from the following:

```
} catch (RemoteException ex) {
   // TODO Auto-generated catch block
   ex.printStackTrace();
}
```

   Change to the following:

```
} catch (Exception ex) {
   //* No logic
}
```

Behind the scenes:

The completed DonateWebRecord processEmployee does the following:

- The EmployeeKey object, needed for calling the EmployeeFacade session EJB, is created and initialized:

  ```
  EmployeeKey primaryKey = new EmployeeKey();
  primaryKey.setEmployeeid(this.getEmployee_Number());
  ```

- The anEmployee object and return message are created:

  ```
  Employee anEmployee = null;
  String message;
  ```

- The getEmployeeByKey method is called by the code inserted by the EJB snippet:

  ```
  //* Insert call to Session bean service method
  EmployeeFacade anEmployeeFacade = createEmployeeFacade();
  try {

     anEmployee = anEmployeeFacade.getEmployeeByKey(primaryKey);

  } catch (Exception ex) {
     //* No logic
  }
  ```

- The returned value of the anEmployee object is evaluated. If it is null, the return message is updated to indicate an error occurred, and a ‘blank’ Employee object is created and assigned to anEmployee. This prevents a NullPointerException when the employeeDetail.jsp page displays the returned object. Otherwise, a successful return message is set.

  ```
  if (anEmployee == null)
  {
     message = "Error: retrieved null record!";
     anEmployee = new Employee();
  } else {
     message = "Record retrieved successfully";
  }
  ```

- The anEmployee object is assigned to the employee_Details object, which is accessible from outside the class using the getters and setters:

  ```
  this.setEmployee_Details(anEmployee);
  ```

- The return message is passed back to the caller:

  ```
  return message;
  ```

9.3.4 Code the processDonation method

This method is used on the donateResult.jsp page to take the input donation_Amount, which was entered on the employeeDetail.jsp page, and call the donateVacation method in the Donate session EJB.

1. Still on the **DonateWebRecord.java** editor, insert a blank line before closing the bracket, and position the cursor on this new line.
2. In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG4: DonateWebRecord processDonation**. This inserts the snippet code into **DonateWebRecord.java** at the current cursor location.

```
public String processDonation() {

   String aString = "";
   int employee_num = this.getEmployee_Details().getEmployeeid();
   String fund_name = "DonationFund";
   int donation_amount = this.getDonation_Amount();
   //* Insert call to Session bean service method

   return aString;
}
```

Figure 9-3 FRAG4: DonateWebRecord processDonation

3. Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux). Note that you should not have any errors at this point.
4. Add the call to the Donate session EJB using a snippet.
   - Select the line after **//* Insert call to Session bean service method**.
   - In the **Snippets** view (lower right), expand the **EJB** category, and double-click **Call a Session bean service method**.
   - At the **Select Reference** pop-up, select **ejb/Donate**, and click **Next**.
   - At the **SelectMethod Values** pop-up, select **donateVacation (int employee_num, String fund_name, int donation_amount),** and click **Finish**.
5. One error remains ("Duplicate local variable of aString"). Fix this by changing this line:

```
String aString = aDonate.donateVacation(
         employee_num, fund_name,donation_amount)
```

   to the following line:

```
aString = aDonate.donateVacation(
         employee_num, fund_name,donation_amount)
```

6. Save and close **DonateWebRecord.java**.

Behind the scenes:

The completed DonateWebRecord processDonation does the following:

- The default (successful) return message is created and initialized:

  ```
  String aString = "";
  ```

- The input parameters, which are needed for the call to the donateVacation method in the Donate session EJB, are initialized from the DonateWebRecord global variables:

  ```
  int employee_num = this.getEmployee_Details().getEmployeeid();
  String fund_name = "DonationFund";
  int donation_amount = this.getDonation_Amount();
  ```

- The donateVacation method is called by the code inserted by the EJB snippet:

  ```
  //* Insert call to Session bean service method
  Donate aDonate = createDonate();
  try {
     aString = aDonate.donateVacation(employee_num, fund_name,
           donation_amount);
  } catch (RemoteException ex) {
     // TODO Auto-generated catch block
     ex.printStackTrace();
  }
  ```

- The return message is passed back to the caller.

  ```
  return aString;
  ```

9.3.5 Work around: Add serviceLocatorMgr.jar to MANIFEST.MF

This jar file (contained in DonateEAR) contains classes that are used by the EJB “Find” method code, which the snippet added. In Rational Application Developer V6.0, this was automatically added to both the classpath and the MANIFEST.MF of the EJB project when running the snippet. In AST V6.1, it is only added to the classpath and not to the MANIFEST.MF. This is a known problem that is fixed in Web Tools Platform 1.5 and will be included in the AST V6.1 update version that includes WTP 1.5.

As a result the donateMethod fails with the following error when invoking the snippet code:

```
CNTR0020E: EJB threw an unexpected (non-declared) exception during
invocation of method "donateVacation" on bean
"BeanId(DonateEAR#DonateEJB2.jar#Donate, null)". Exception data:
java.lang.NoClassDefFoundError:
com.ibm.etools.service.locator.ServiceLocatorManager
```

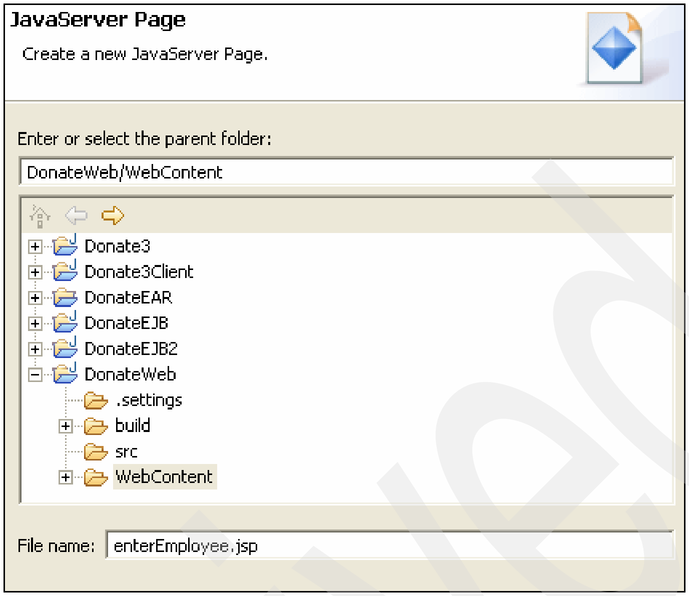
1. From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **META-INF** → **MANIFEST.MF** and double-click to open.
2. From the resulting **Jar Dependencies** editor, scroll down to the **Dependencies** section, and select **serviceLocatorMgr.jar**.
3. Save and close.

9.4 Create and test the Web pages

In this section you create and test the enterEmployee, displayEmployee, and donateResult JSP pages.

9.4.1 Create the JSP pages

1. Create the **enterEmployee.jsp** page by performing the following:
   - From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **WebContent**. Right-click **New** → **JSP**.
   - At the **JavaServer Page** pop-up, set **File name** to **enterEmployee.jsp**, and click **Finish**.



JavaServer Page

Create a new JavaServer Page.

Enter or select the parent folder:

DonateWeb/WebContent

```
+ Donate3
+ Donate3Client
+ DonateEAR
+ DonateEJB
+ DonateEJB2
- DonateWeb
    .settings
  + build
    src
  + WebContent
```

File name: enterEmployee.jsp

2. Create the **employeeDetail.jsp** page by performing the following actions:
   - From the **Project Explorer**, select **Dynamic Web. Projects** → **DonateWeb** → **WebContent**. Right-click **New** → **JSP**.
   - At the **JavaServer Page** pop-up, set **File name** to **employeeDetail.jsp**, and click **Finish**.
3. Create the **donateResult.jsp** page, and perform the following actions:
   - From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **WebContent**, and right-click **New** → **JSP**.
   - At the **JavaServer Page** pop-up, set **File name** to **donateResult.jsp**, and click **Finish**.

9.4.2 Complete and test enterEmployee.jsp

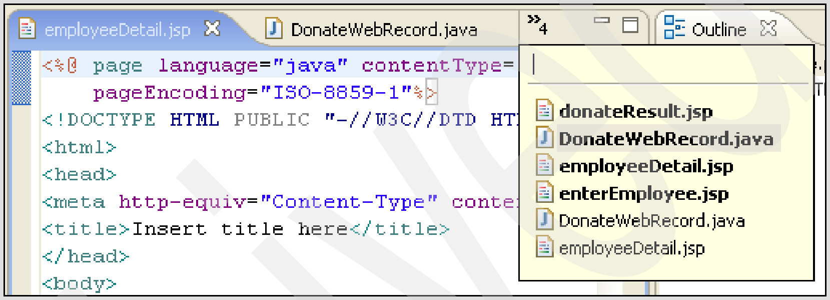
As discussed previously, AST V6.1 does not provide visual editors for JSP or HTML pages. Therefore, this document provides all JSPs through snippets. Reference section 9.5, "Explore!" on page 243 for further discussion on visual editor capabilities in Rational Application Developer.

1. Switch to the open **enterEmployee.jsp** editor.

Off course?

If the editor is not open, simply select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **enterEmployee.jsp**, and double-click to open.

If it is still not visible, perhaps the editor is open but is not currently displayed. This may occur if you have more editors and views open than can be displayed on the title bar. If this occurs, you will have an overflow icon [»4]on the right edge of the title bar. Left-click this icon to see the list of other opened editors and view, and then select to open.



```
<%@ page language="java" contentType=
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HT
<html>
<head>
<meta http-equiv="Content-Type" conte
<title>Insert title here</title>
</head>
<body>
```

2. Change the title line:

```
From: <title>Insert title here</title>
To:   <title>enterEmployee</title>
```

3. Insert the content by performing the following actions:

   – Place the cursor on the blank line between <body> and </body>.

   – In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG5: enterEmployee jsp**. This inserts the snippet code into **enterEmployee.jsp** at the current cursor location.

```
<h1>Enter Employee Information</h1>
<FORM METHOD=POST  ACTION="employeeDetail.jsp">
<table>
<tr>
<td align="left">Employee_Number:</td>
<td>
<INPUT TYPE=TEXT NAME=employee_Number SIZE=20/>
<INPUT TYPE=SUBMIT VALUE="Lookup" />
</td>
</tr>
</table>
</FORM>
```

Figure 9-4   FRAG5: enterEmployee jsp

Behind the scenes:

- Display the page title:

  ```
  <h1>Enter Employee Information</h1>
  ```

- Define the form action that submits to employeeDetail.jsp when the user clicks the **Submit** button:

  ```
  <FORM METHOD=POST  ACTION="employeeDetail.jsp">
  ```

- Display the first cell in the column (containing the Employee_number: text):

  ```
  <table>
  <tr>
  <td align="left">Employee_Number:</td>
  ```

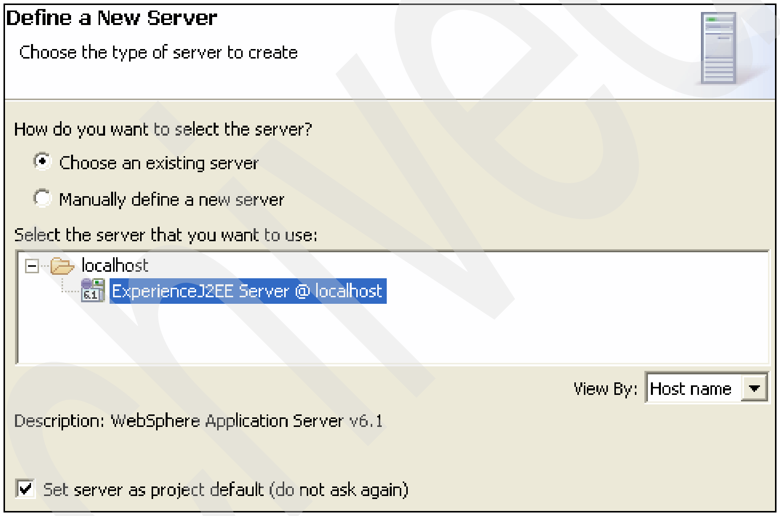
- Display the input field (in the second column). The value that the user enters is mapped to the employee_Number property when the user clicks the **Submit** button:

  ```
  <td>
  <INPUT TYPE=TEXT NAME=employee_Number SIZE=20/>
  ```

- Display the **Submit** button, and close out the cell/row/table/form definitions:

  ```
  <INPUT TYPE=SUBMIT VALUE="Lookup"/>
  </td>
  </tr>
  </table>
  </FORM>
  ```

4. Save and close **enterEmployee.jsp**.
5. Test **enterEmployee.jsp**, using the following steps:
   - Select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **enterEmployee.jsp**, and right-click **Run-As** → **Run on Server**.
   - If you receive the **Define a New Server** pop-up, perform the following actions:
     - Select **choose an existing server**.
     - Select **localhost** → **ExperienceJ2EE Server @ localhost**.
     - Select **Set server as project default (do not ask again)**.
     - Click **Finish**.



Define a New Server

Choose the type of server to create

How do you want to select the server?

● Choose an existing server

○ Manually define a new server

Select the server that you want to use:

localhost
ExperienceJ2EE Server @ localhost

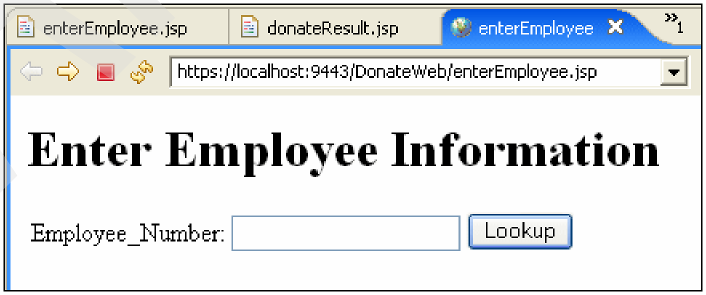
View By: Host name

Description: WebSphere Application Server v6.1

☑ Set server as project default (do not ask again)

The AST V6.1 internal Web browser should open to the enterEmployee page:

`https://localhost:9443/DonateWeb/enterEmployee.jsp`



https://localhost:9443/DonateWeb/enterEmployee.jsp

Enter Employee Information

Employee_Number:

Behind the scenes:

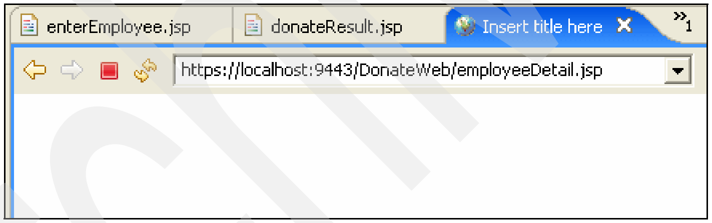
Note that the URL defaults to HTTPS (over port 9443) instead of HTTP (over port 9080):

- AST V6.1 defaults to HTTPS because J2EE administrative security is enabled.
- AST V6.1 uses port 9443 for HTTPS because that is the default value you used for the HTTPS transport port on the Port Values Assignment pop-up in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51.

Note that you can manually change the URL to use HTTP over the default HTTP transport port of 9080:

```
http://localhost:9080/DonateWeb/enterEmployee.jsp
```

– Test the navigation to **employeeDetail.jsp** by clicking **Lookup** (no value is needed at this point). The browser should transition to the **employeeDetail.jsp** page but shows no content because you have not edited that file yet.



https://localhost:9443/DonateWeb/employeeDetail.jsp

9.4.3 Complete and test employeeDetail.jsp

1. Switch to the open **employeeDetail.jsp** editor.

2. Change the title line:

```
From: <title>Insert title here</title>
To:   <title>employeeDetail</title>
```

3. Insert the content using the following steps:

   – Place the cursor on the blank line between <body> and </body>.

   – In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG6: employeeDetail jsp.** This inserts the snippet code into **employeeDetail.jsp** at the current cursor location.

```
<jsp:useBean id="record" class="donate.web.DonateWebRecord"
scope="session"/>
<jsp:setProperty name="record" property="employee_Number"/>
<h1>Employee Detail</h1>
<%=record.processEmployee() %><BR>
<BR>
<table>
<tr>
   <td align="left">Employeeid:</td>
   <td><%= record.getEmployee_Details().getEmployeeid() %></td></tr>
<tr>
   <td align="left">Firstname:</td>
   <td><%= record.getEmployee_Details().getFirstname() %></td></tr>
<tr>
   <td align="left">Middlename:</td>
   <td><%= record.getEmployee_Details().getMiddlename() %></td></tr>
<tr>
   <td align="left">Lastname:</td>
   <td><%= record.getEmployee_Details().getLastname() %></td></tr>
<tr>
   <td  width="" align="left">Salary:</td>
   <td><%= record.getEmployee_Details().getSalary() %></td></tr>
<tr>
   <td align="left">Vacamount:</td>
   <td><%= record.getEmployee_Details().getVacamount() %></td>
</tr>
</table>
<P>
Would you like to donate some of your vacation to the Donation fund
to be used by those with family emergencies who need additional
vacation? If yes, enter the amount you wish to donate and press
Submit. Thank you for your generosity!
</P>
<FORM METHOD=POST ACTION="donateResult.jsp">
<table>
<tr><td align="left">Donation Amount:</td>
<td><INPUT TYPE=TEXT NAME=donation_Amount SIZE=20/>
<INPUT TYPE=SUBMIT VALUE="Donate"/>
</td></tr>
</table>
</FORM>
```

Figure 9-5 FRAG6: employeeDetail jsp

Behind the scenes:

- Initialize the DonateWebRecord bean by setting the scope to Session (meaning for all pages) and setting the identifier to record:

```
<jsp:useBean id="record" class="donate.web.DonateWebRecord"
scope="session"/>
```

- Map between the employee_Number value input from enterEmployee.jsp, and the employeee_Number in record (the DonateWebRecord bean):

```
<jsp:setProperty name="record" property="employee_Number"/>
```

- Display the page title:

```
<h1>Employee Detail</h1>
```

- Execute the processEmployee method to invoke the Employee session EJB getEmployeeByKey method (and within the processEmployee method it assigns the returned Employee object to the employee_Details variable within the DonateWebRecord bean):

```
<%=record.processEmployee() %><BR>
```

- Display the resulting employee_Details information. Note that some of the table rows are omitted here in the interest of brevity.

```
<BR>
<table>
<tr>
   <td align="left">Employeeid:</td>
   <td><%= record.getEmployee_Details().getEmployeeid()
%></td></tr>
<tr>
   <td align="left">Firstname:</td>
   <td><%= record.getEmployee_Details().getFirstname()
%></td></tr>
...
...
<tr>
   <td align="left">Vacamount:</td>
   <td><%= record.getEmployee_Details().getVacamount() %></td>
</tr>
</table>
```

- Display the text asking the user to donate vacation:

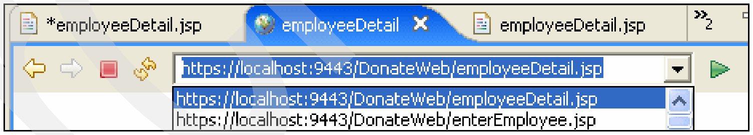
  ```
  <P>
  Would you like to donate some of your vacation to the Donation
  fund to be used by those with family emergencies who need
  additional vacation? If yes, enter the amount you wish to
  donate and press Submit. Thank you for your generosity!
  </P>
  ```

- Define the form, input field, and **Submit** button required to permit the user to enter a donation amount. Click **Submit** ("Donate"), and then have the page transition to donateResult.jsp:

  ```
  <FORM METHOD=POST ACTION="donateResult.jsp">
  <table>
  <tr><td align="left">Donation Amount:</td>
  <td><INPUT TYPE=TEXT NAME=donation_Amount SIZE=20/>
  <INPUT TYPE=SUBMIT VALUE="Donate"/>
  </td></tr>
  </table>
  </FORM>
  ```

4. Save and close **employeeDetail.jsp**.
5. Test **employeeDetail.jsp** by performing the following actions:

   – Switch to the AST V6.1 Web browser, and use the Web history pull-down to navigate back to **enterEmployee.jsp**. This is intentional: Start with enterEmployee, and then navigate to employeeDetail.



https://localhost:9443/DonateWeb/employeeDetail.jsp

https://localhost:9443/DonateWeb/employeeDetail.jsp
https://localhost:9443/DonateWeb/enterEmployee.jsp

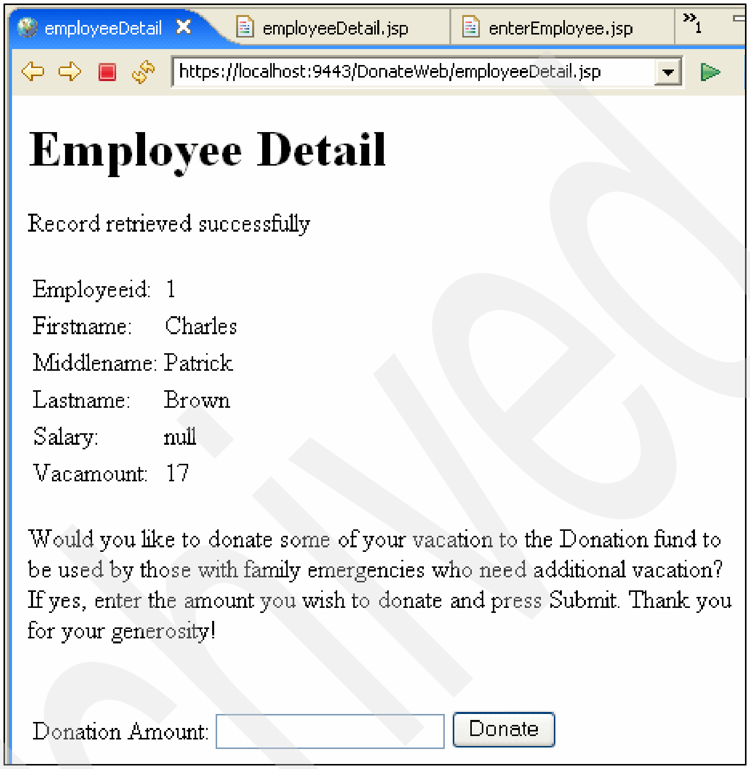
Off course?

If the AST V6.1 Web browser is not open, select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **enterEmployee.jsp**, and right-click **Run-As** → **Run on Server**.

   – The AST V6.1 internal Web browser should open to the enterEmployee page:

   ```
   https://localhost:9443/DonateWeb/enterEmployee.jsp
   ```

- Test the navigation to **employeeDetail.jsp** by entering an **Employee_Number** of **1** and then clicking **Lookup**. The browser should transition to the **employeeDetail.jsp** page and display the correct Employee information.



Employee Detail

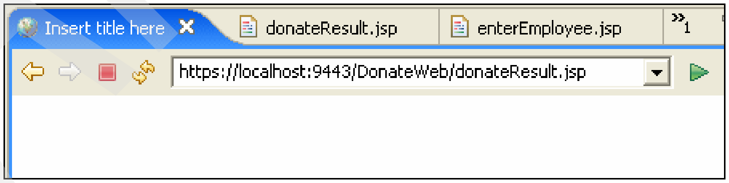
Record retrieved successfully

Employeeid: 1
Firstname: Charles
Middlename: Patrick
Lastname: Brown
Salary: null
Vacamount: 17

Would you like to donate some of your vacation to the Donation fund to be used by those with family emergencies who need additional vacation? If yes, enter the amount you wish to donate and press Submit. Thank you for your generosity!

Donation Amount: Donate

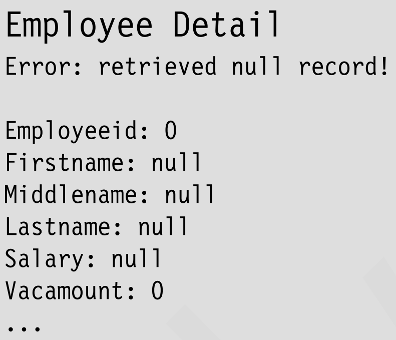
- Test the navigation to **donateResult.jsp** by clicking **Donate** (no value is needed at this point). The browser should transition to the **donateResult.jsp page** but shows no content because you have not edited that file yet.



Extra credit:

Try an error condition:

- Use the browser's back icon [⇦] to return to enterEmployee.jsp and lookup an invalid employee number (such as 99). employeeDetail.jsp should indicate that a null record was returned:



```
Employee Detail
Error: retrieved null record!

Employeeid: 0
Firstname: null
Middlename: null
Lastname: null
Salary: null
Vacamount: 0
...
```

- Note that there is no indication of what the specific error was. The error handling in the EmployeFacade getEmployeeByKey and the DonateWebRecord processEmployee methods are very basic and can only indicate success or failure.

9.4.4 Complete and test donateResult.jsp

1. Switch to the open **donateResult.jsp** editor.
2. Change the title line:

```
From: <title>Insert title here</title>
To:   <title>Donation Result</title>
```

3. Insert the content by performing the following actions:
   – Place the cursor on the blank line between <body> and </body>.
   – In the **Snippets** view (lower right), expand the **Experience J2EE** category and double-click **FRAG7: donateResult jsp**. This inserts the snippet code into **donateResult.jsp** at the current cursor location.

```
<jsp:useBean id="record" class="donate.web.DonateWebRecord"
scope="session"/>
<jsp:setProperty name="record" property="donation_Amount"/>
<h1>Donation Result</h1>
<br>
<%=record.processDonation()%
```

Figure 9-6 FRAG7: donateResult jsp

Behind the scenes:

- Reuse the same DonateWebRecord bean by specifying an id of record that was initialized in employee_Detail.jsp:

  ```
  <jsp:useBean id="record" class="donate.web.DonateWebRecord"
  scope="session"/>
  ```

- Map between the donation_Amount value input from employeeDetail.jsp and the donation_Amount variable in record (the DonateWebRecord bean):

  ```
  <jsp:setProperty name="record" property="donation_Amount"/>
  ```

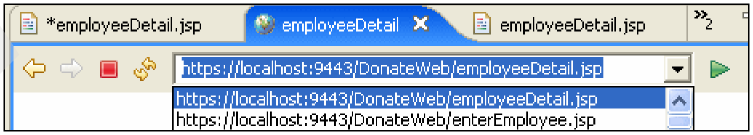
- Display the page title:

  ```
  <h1>Donation Result</h1>
  ```

- Execute the processDonation method to invoke the donateVacation method and to print the return message on the browser:

  ```
  <br>
  <%=record.processDonation()%>
  ```

4. Save and close **donateResult.jsp**.
5. Test **donateResult.jsp** by performing the following actions:
   – Switch to the AST V6.1 Web browser, and use the Web history pull-down to navigate back to **enterEmployee.jsp**. This is intentional: Start with enterEmployee (and then navigate to employeeDetail and then to donateResult).



https://localhost:9443/DonateWeb/employeeDetail.jsp

https://localhost:9443/DonateWeb/employeeDetail.jsp

https://localhost:9443/DonateWeb/enterEmployee.jsp

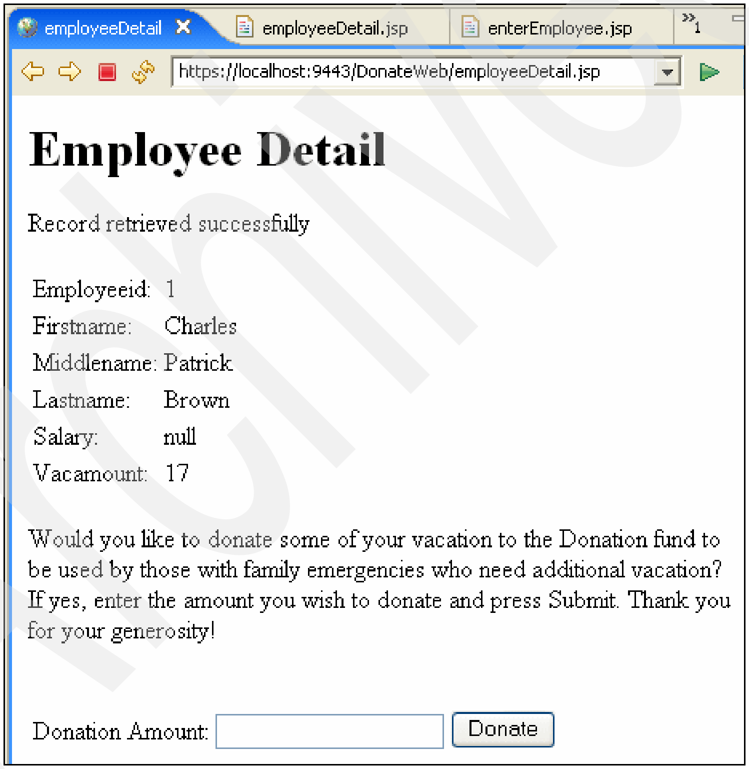
Off course?

If the AST V6.1 Web browser is not open, select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **enterEmployee.jsp**, and right-click **Run-As** → **Run on Server**.

– The AST V6.1 internal Web browser should open to the enterEmployee page:

```
https://localhost:9443/DonateWeb/enterEmployee.jsp
```

– Test the navigation to **employeeDetail.jsp** by entering an **Employee_Number** of **1** and then clicking **Lookup**. The browser should transition to the **employeeDetail.jsp** page and display the correct Employee information.



Employee Detail

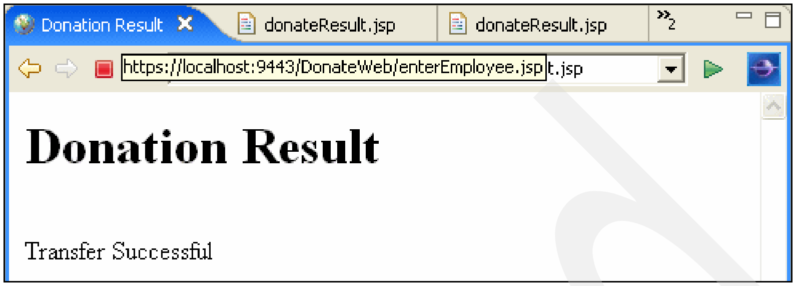
Record retrieved successfully

Employeeid: 1
Firstname: Charles
Middlename: Patrick
Lastname: Brown
Salary: null
Vacamount: 17

Would you like to donate some of your vacation to the Donation fund to be used by those with family emergencies who need additional vacation? If yes, enter the amount you wish to donate and press Submit. Thank you for your generosity!

Donation Amount:

- Test the navigation to **donateResult.jsp** by entering a **Donation Amount** of **1** and then clicking **Donate**. The browser should transition to the **donateResult.jsp** page and show the donation result:



Donation Result

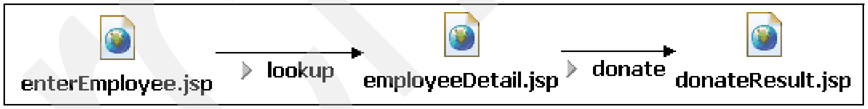
Transfer Successful

6. Close all open editors, viewers, and Web browsers.

9.5 Explore!

As described in the learn section, JSF (also called Faces) provides a development environment and runtime for creating a Web application consisting of one or more JSP pages. WTP V2.0 is slated to contain a JSF development component.

The following is a screen capture from a Rational Application Developer V6.0 JSF visual composition page (called the Web Diagram) that was used to create a JSF equivalent of the Web front end that you developed in this chapter.



In general, a JSF Web application consists of the following components:

- JSF Web pages, which are JSPs with included JSF tag libraries.
- Pagecode managed beans, which provide backing support (such as input validation and action processing) to the associated Web page. Not shown in the screen capture.

  JSF provides a default Pagecode class where you can include all common functions and then unique Pagecode classes per Web page that extend this default class.

  The pagecode classes are defined in a text file (faces-config.xml file).
- Managed beans, which contain the information that is shared between the pages. Not shown on the screen capture.

JSF Managed beans are data objects that are stored and retrieved from the Session store. These are defined visually, and the definition is stored in a text file (faces-config.xml file):

```
<managed-bean>
   <managed-bean-name>donateRecord</managed-bean-name>
   <managed-bean-class>donate.web.DonateWebRecord
      </managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

JSF tooling automatically adds getters and setters to the Pagecode managed beans, and the developer can use these methods to access the object from the Session store.

- Navigation rules provide the linkage between the pages based on outcomes specified by actions run in the PageCode Managed beans. The arrows labeled lookup and donate in the screen capture above represent navigation rules.

  For example, when the user presses Submit on the enterEmployee.jsp page:

  – The action is directed to the JSF controlling servlet, which then invokes the enterEmployee.jsp Pagecode managed bean.

  – The Pagecode managed bean invokes the Employee Facade entity EJB to retrieve the specified object, stores it in a JSF Managed bean, and then returns the string “lookup”.

  The JSF controlling servlet looks up this action from the faces-config.xml and routes to employeeDetail.jsp to display the result:

```
<from-view-id>/enterEmployee.jsp</from-view-id>
<navigation-case>
   <from-outcome>lookup</from-outcome>
   <to-view-id>/employeeDetail.jsp</to-view-id>
   <redirect />
</navigation-case>
```

Additional explore resources are available at the following Web address:

- The J2EE 1.4 Tutorial: Chapter 17: JavaServer Faces Technology:
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
- IBM developerWorks: JSF for nonbelievers: Clearing the FUD about JSF:
  http://www.ibm.com/developerworks/java/library/j-jsf1/index.html
- IBM Education Assistant: Developing Web applications with JavaServer Faces:
  http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.rad_v6/rad/6.0/JSF.htm

10

Create the application client

In this chapter you create a simple non-graphical, meaning text interaction versus graphical interaction, J2EE application client that provides a *thick client* to the Donate session EJB.

10.1 Learn!

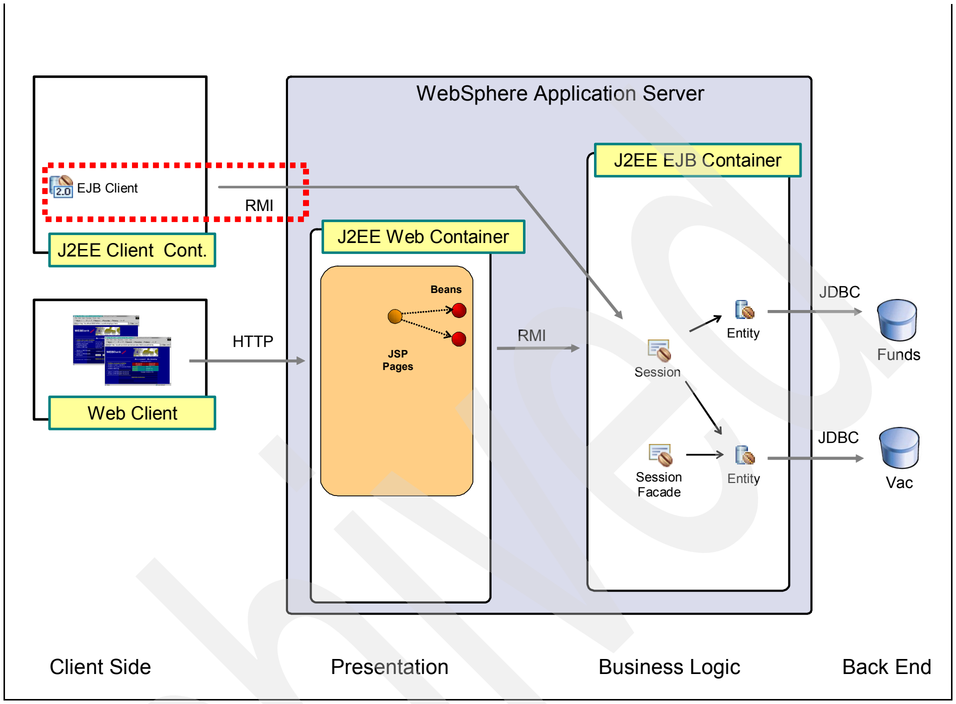


Figure 10-1 Donate application: J2EE Application Client

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 the following:

- 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, J2EE provides a *thick client* called the J2EE 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 J2EE server-side environment using resources such as JNDI, EJBs, JDBC, and JMS.

The example used in this chapter is a simple, non-graphical client. However, most real-world application clients are implemented using a graphical display capability such as Swing, AWT, or SWT.

There are no specific learn resources for this chapter.

10.2 Create the application client project

In this section you create the application client project that will contain the Java main class and any other required classpath definitions/artifacts.

1. From the **Project Explorer**, select **Application Client Projects**, and right-click **New** → **Application Client Project**.
2. At the **Application Client module** pop-up (first pop-up with this name), complete the following fields:
   - **Project Name**: **DonateClient**
   - **Target runtime**: **WebSphere Application Server v6.1**
   - Select **Add Module to an EAR project**
   - **EAR Project Name**: **DonateEAR**
3. Click **Next**.
4. At the **Select Project Facets** pop-up, click **Next**.
5. At the **Application Client module** pop-up (second pop-up with this name), select **Create a default Main** class, and click **Finish**.

10.3 Create the EJB reference

The main method accesses the Donate session EJB. Per J2EE best practices, you access this by using an EJB reference, instead of a direct JNDI lookup.

1. From the **Project Explorer**, select **Application Client Projects** → **DonateClient** → **DonateClient**, and then double-click to open the deployment descriptor.

2. From the resulting **Client Deployment Descriptor** editor, switch to the **References** tab.
3. Add the Donate EJB reference by performing the following actions:
   - Click **Add...**
   - At the resulting **Reference** pop-up, select **EJB reference**, and click **Next**.
   - At the **EJB Reference** pop-up perform the following actions:
     - Select **Enterprise Beans in the workspace**.
     - Select **DonateEAR** → **DonateEJB2** → **Donate**. Note that the **Name** field is automatically populated with **ejb/Donate**.
     - Set **Ref Type** to **Remote**.
   - Click **Finish**.
4. Save and close the client deployment descriptor.

10.4 Code the main method

As with any standalone Java application, a main method is required to start the application. Use the following instructions to code the main method:

1. Select **Application Client Projects** → **DonateClient** → **appClientModule** → **(default package)** → **Main.java**, and then double-click to open in an editor. Note that the class currently contains the following main method:

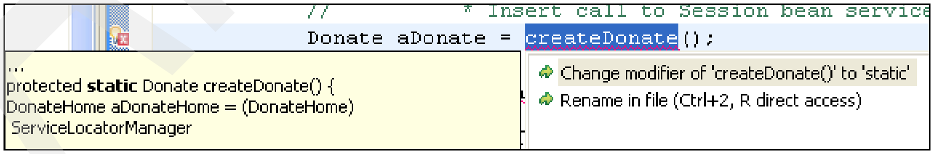
```
public static void main(String[] args) {
     // TODO Auto-generated method stub
}
```

2. In the **Main.java** editor, replace the line // TODO Auto-generated method stub (shown in step 1) with a blank line, and position the cursor on this new blank line.
3. In the **Snippets** view (lower right), expand the **Experience J2EE** category and double-click **FRAG8: DonateClient**.

```
if (args.length == 3) {
   int employee_num = (new Integer(args[0])).intValue();
   String fund_name = args[1];
   int donation_amount = (new Integer(args[2])).intValue();
   String aString = null;

   System.out.println("Employee Number = " + employee_num);
   System.out.println("Fund            = " + fund_name);
   System.out.println("Donation Amount = " + donation_amount);
   //       * Insert call to Session bean service method

      System.out.println("Donation result = " + aString);
} else {
   System.out.println("Program requires 3 command line arguments.");
}
```

Figure 10-2 FRAG8: DonateClient

4. Select the line after **//* Insert call to Session bean service method**.
5. In the **Snippets** view (lower right), expand the **EJB** category, and double-click **Call a Session bean service method**.
   - At the **Select Reference** pop-up, select **ejb/Donate**, and click **Next**.
   - At the **Select Method** pop-up, select **donationVacation(int employee_num, String fund_name, int donation_amount)**, and click **Finish**.
6. Adjust the generated code:
   - At the error for the line Donate aDonate = createDonate() that indicates **Cannot make a static reference to the non-static method createDonate() from the type Main**, left-click the quick fix icon [ ] in the left hand margin and accept the quick fix recommendation to change the modifier of createDonate() to static.



```
//                * Insert call to Session bean service
        Donate aDonate = createDonate();
```

```
...
protected static Donate createDonate() {
DonateHome aDonateHome = (DonateHome)
 ServiceLocatorManager
```

Change modifier of 'createDonate()' to 'static'
Rename in file (Ctrl+2, R direct access)

   - At the error for the line String aString = aDonate... that indicates Duplicate local variable aString, change the line from:

```
String aString = aDonate.donateVacation(employee_num,
      fund_name, donation_amount);
```

to:

```
aString = aDonate.donateVacation(employee_num,
      fund_name, donation_amount);
```

7. Save and close **Main.java**.

Behind the scenes:

- Verify that the appropriate arguments were passed to the main method:

```
if (args.length == 3) {
   int employee_num = (new Integer(args[0])).intValue();
   String fund_name = args[1];
   int donation_amount = (new Integer(args[2])).intValue();
   String aString = null;

   System.out.println("Employee Number = " + employee_num);
   System.out.println("Fund            = " + fund_name);
   System.out.println("Donation Amount = " + donation_amount);
...
...
} else {
   System.out.println("Program requires 3 command line
arguments.");
```

- Call the Session bean service method—added by the wizard and then modified:

```
// * Insert call to Session bean service method
Donate aDonate = createDonate();
try {
   aString = aDonate.donateVacation(employee_num,
         fund_name, donation_amount);
} catch (RemoteException ex) {
   // TODO Auto-generated catch block
   ex.printStackTrace();
}
```

- Print out the results:

```
System.out.println("Donation result = " + aString);
```

10.5 Test the application client

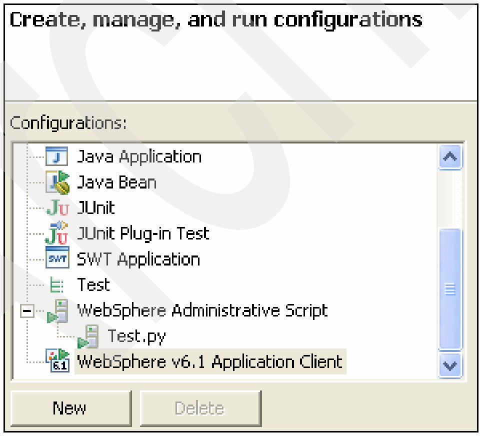
J2EE application clients require a configured runtime environment called the Client container. This is similar to the Web container used for Servlets and JSPs and the EJB container used for EJBs. With WAS V6.1 the following applies:

- 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).

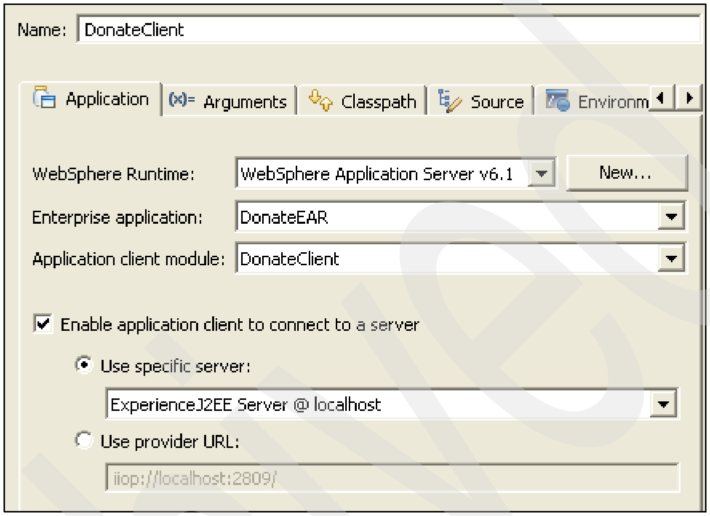
This development environment scenario, using the AST V6.1 tooling and the WAS V6.1 runtime, is such a rare situation—Running an Application Client on the same system as the Application Server.

1. From the **Project Explorer**, select **Application Client Projects** → **DonateClient**, and from the action bar select **Run** → **Run…**.
2. At the **Create, Manage, and run Configurations** pop-up perform the following actions:
   - On the left, under **Configurations**, select **WebSphere v6.1 Application Client**, and select **New**.


Create, manage, and run configurations
Configurations:
Java Application
Java Bean
JUnit
JUnit Plug-in Test
SWT Application
Test
WebSphere Administrative Script
Test.py
WebSphere v6.1 Application Client
New
Delete

   - On the resulting right pane of the **Application** tab, complete the following fields:
     - **Name**: **DonateClient**
     - **WebSphere Runtime**: **WebSphere Application Server V6.1**

- **Enterprise application**: **DonateEAR**
- **Application client module**: **DonateClient**
- Select **Enable application client to connect to a server**.
- Select **Use specific serve**r, and then select **ExperienceJ2EE Server @localhost**.



Name: DonateClient

WebSphere Runtime: WebSphere Application Server v6.1

Enterprise application: DonateEAR

Application client module: DonateClient

☑ Enable application client to connect to a server

● Use specific server:

ExperienceJ2EE Server @ localhost

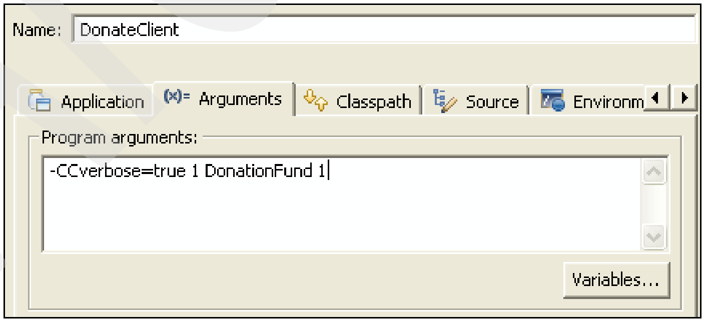
○ Use provider URL:

iiop://localhost:2809/

– On the resulting right pane on the **Arguments** tab, perform the following actions:

  - 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:

```
-CCverbose=true 1 DonationFund 1
```



Name: DonateClient

Program arguments:

-CCverbose=true 1 DonationFund 1

– Click **Apply** (in the lower right of the pop-up) to save the changes from both the Application and Arguments tab.

– Click **Run** to execute the Application Client.

AST V6.1 should automatically switch to the **Console** view in the lower-right pane, displaying the output of the Application Client.

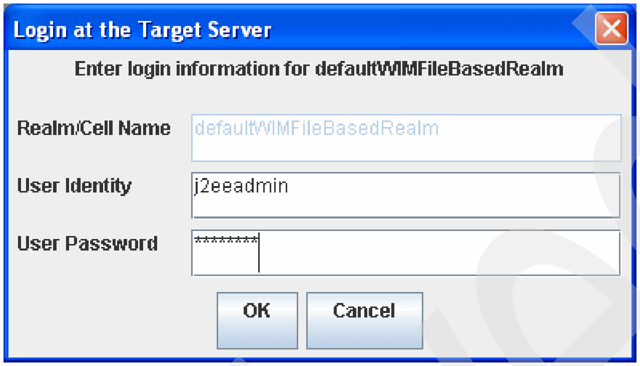
Off course?

If you are utilizing the external Web browser, you may notice these messages or prompts in the **Console** view the first time you run the Application Client:

```
*** SSL SIGNER EXCHANGE PROMPT ***
SSL signer from target host 127.0.0.1 is not found in trust
store /opt/IBM/AppServer/profiles/AppSrv01/etc/trust.p12.

Here is the signer information (verify the digest value matches
what is displayed at the server):

Subject DN:    CN=localhost.localdomain, O=IBM, C=US
Issuer DN:     CN=localhost.localdomain, O=IBM, C=US
Serial number: 1156274410
Expires:       Wed Aug 22 15:20:10 EDT 2007
SHA-1 Digest:
4B:FC:2E:61:31:CF:4E:12:5F:FA:08:AA:14:23:B7:ED:E8:06:D4:99
MD5 Digest:    41:3F:53:4F:0F:F8:2F:C6:FE:21:1F:48:54:FB:16:1E

Add signer to the trust store now? (y/n) y
```

Enter Y at the prompt, and the execution should continue as normal.

3. Since administrative security is enabled, a pop-up appears when the Application Client attempts to query the JNDI information. Enter the j2eeadmin WAS V6.1 administrative id and password (defined in “3.5 Create the ExperienceJ2EE application server profile” on page 51).
4. Click **OK**.



Login at the Target Server

Enter login information for defaultWIMFileBasedRealm

Realm/Cell Name: defaultWIMFileBasedRealm

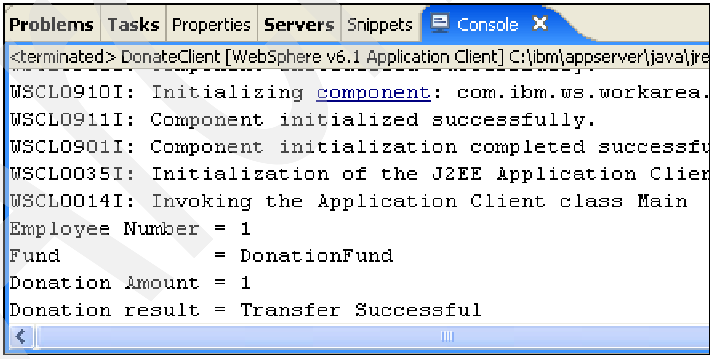
User Identity: j2eeadmin

User Password: *********

Off course?

Sometimes this pop-up may be obscured, so if you do not see it, and the **Console** view appears to be stopped at Donation Amount = 1, try minimizing some other windows until you see this.

5. At the **Console** view, verify that the Application Client completed successfully.



<terminated> DonateClient [WebSphere v6.1 Application Client] C:\ibm\appserver\java\jre

```
WSCL0910I: Initializing component: com.ibm.ws.workarea.
WSCL0911I: Component initialized successfully.
WSCL0901I: Component initialization completed successfu
WSCL0035I: Initialization of the J2EE Application Clien
WSCL0014I: Invoking the Application Client class Main
Employee Number = 1
Fund            = DonationFund
Donation Amount = 1
Donation result = Transfer Successful
```

Off course?

If you do not see the DonateClient output in the **Console** view, select the Donate Client output from the Display Selected Console icon pull-down [🖥 ▾].

The **Console** view automatically switches to the most recent output. So it is very possible that a message on the ExperienceJ2EE Server @ localhost could cause a switch.

10.6 Explore!

The application client used here is a J2EE (application) client, and it provides the full functionality as defined by the J2EE specification—at a price. The client system must have a J2EE client container, in this case provided by the WebSphere Application Server client feature, running on the same J2SE as the J2EE application server, and must be started using the (WebSphere) **`launchclient`** command.

However, you may find that these requirements are too restrictive for your situation. For example, you may have an existing Java application that utilizes a different J2SE and you wish to call an EJB on WAS V6.1. You would not meet the Application Client requirements of starting using the **`launchclient`** command (versus a Java call from an existing JVM) and using the WAS V6.1 J2SE version (versus using the existing different J2SE). Therefore, WAS V6.1 provides the following non-J2EE complaint application clients that you can use as an alternative:

- Thin client
- Pluggable client
- Applet client (Windows only)
- ActiveX® client

Additional explore resources are available at the following Web address:

- WAS V6.1 Infocenter: Client applications:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/welc6tech_cli.html`

- *WebSphere Application Server V6 System Management & Configuration Handbook*, SG24-6451:

  `http://www.redbooks.ibm.com/redbooks/pdfs/sg246451.pdf`

11

Implement core security

In this chapter you secure Web and J2EE application client access to the application.

11.1 Learn!

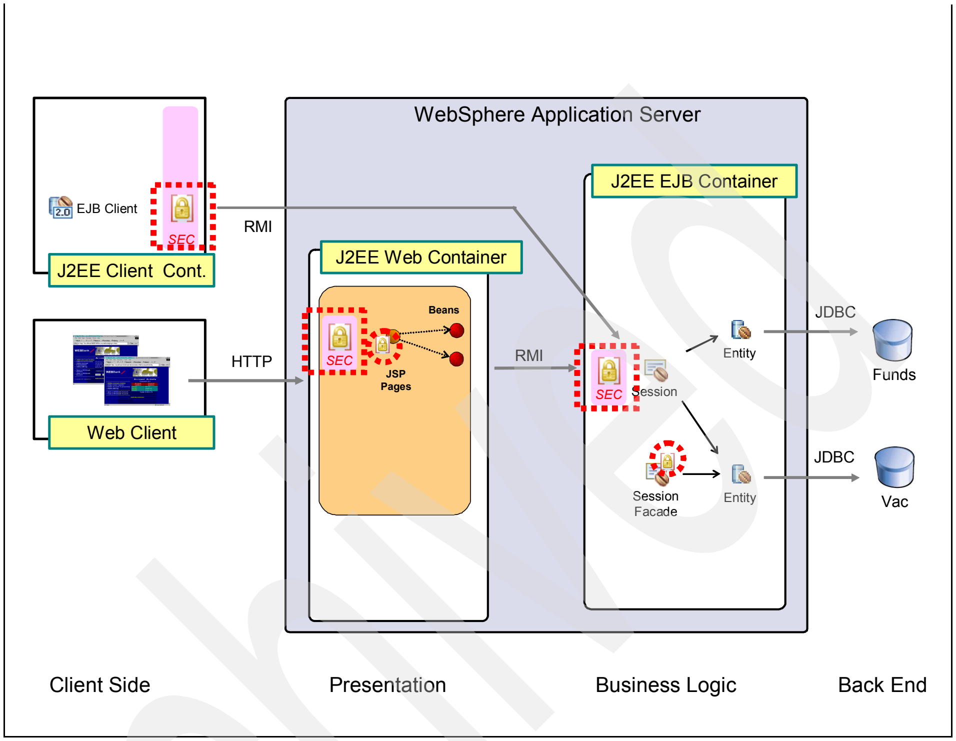


Figure 11-1 Donate application: J2EE application security

J2EE contains a multi-level security architecture for applications. Preceding chapters touched on one layer—J2C authentication data— and this chapter touches on the following:

- User registry, which defines the valid users and groups within a J2EE environment
- Application role based security, which restricts access to J2EE 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
- Java2 Security

- Java Authorization Contract for Containers (JACC)

Additional learn resources include the following:

- The J2EE 1.4 Tutorial: Chapter 32: Security:

  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

- *WebSphere Application Server V6.1 Security Handbook*, SG24-6316:

  http://www.redbooks.ibm.com/abstracts/sg246316.html

- *WebSphere Application Server V6.1 Security Handbook*, SG24-6316, draft available at:

  http://www.redbooks.ibm.com/redpieces/abstracts/sg246316.html

  Expected redbook publish date: 12/20/2006

11.1.1 User registry

Any J2EE application server that utilizes application role-based security or administrative security (which will be discussed in the subsequent sections) requires a mechanism for authenticating users and groups.

The J2EE mechanism for this type of security is a user registry, and WAS V6.1 supports the following:

- Local operating system user registry
- Lightweight Directory Access Protocol (LDAP) user registry
- File based user registry (new with V6.1)
- Custom user registry (java class created by the user)

Access to a secured J2EE resource results in a request against the user registry.

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 J2EE authentication mechanisms.

11.1.2 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*. J2EE 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.

J2EE has two forms of role based security:

- Declarative security, where the security constraints limiting access are defined in the deployment descriptors, and the J2EE runtime is the mechanism that manages access to the J2EE 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 his or her role, the user may only see a subset of the Employee data.

You must configure a user registry before you activate application role based security. In this case, the file-based user registry is already configured because you created the profile with administrative security already enabled, which automatically configured the user registry.

11.1.3 Administrative security

All J2EE application servers should have secure access to their administrative interfaces (such as the Admin Console GUI and **`wsadmin`** command for WAS V6.1), and these are usually implemented using the J2EE application role based security model.

In the case of WAS V6.1, this implementation allows the following predefined roles: Administrator, Operator, Configurator, Monitor, iscadmins, and adminsecuritymanager.

Starting with WAS V6.1, 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.

11.2 Preliminary security setup

In this section you create the users and groups for use in demonstrating application role based security:

Group	User members
DUSERS	DCBROWN, DNARMSTRONG
DMANAGERS	DNARMSTRONG
<no group>	DGADAMS

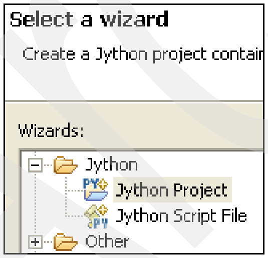
WAS V6.1 allows you to define the users in the file-based registry by manually editing the file, through the Admin Console, or by using the `wsadmin` command.

The following sections add the users and groups using the `wsadmin` command.

11.2.1 Create and configure the Jython project and script

Jython is the recommended scripting language for WAS V6.1 administration. In this section you use AST V6.1 to create a Jython project, and then create the Jython script that adds the users and groups.

1. From the **Project Explorer**, select **Other Projects**, and right-click **New** → **Other...**
2. At the **Select a wizard** pop-up, select **Jython** → **Jython Project**, and click **Next**.



Select a wizard

Create a Jython project contai

3. At the **Create a Jython project** pop-up, set the **Project name** to **DonateJython**, and click **Finish**.
4. From the **Project Explorer,** select **Other Projects** → **DonateJython**, and right-click **New** → **Other...**
5. At the **Select a wizard** pop-up, select **Jython** → **Jython Script File**, and click **Next**.

6. At the **Create a Jython script file** pop-up, set the **File name** to **AddGroupsUsers.py**, and click **Finish**.
7. Switch to the open **AddGroupsUsers.py** editor (**Other Projects** → **DonateJython** → **AddGroupsUsers.py**):
   - Select the cursor on the last blank line.
   - In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click on **FRAG9: 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 xxxxxxx
   -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')
```

Figure 11-2 FRAG9: AddGroupsUsers

   - 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:

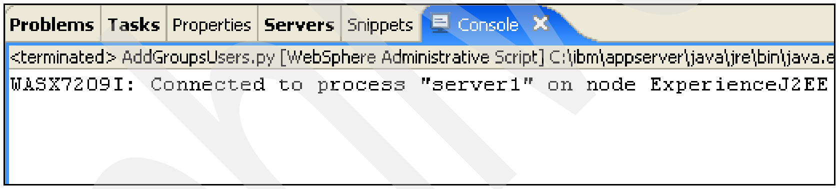
```
AdminTask.createUser('-uid DCBROWN -password mypassword
   -confirmPassword mypassword -cn DCBROWN -sn DCBROWN')
```

8. Save and close **AddGroupsUsers.py**.

11.2.2 Run the Jython script

In this section you execute the Jython script to add the users and groups.

1. From the **Project Explorer,** select **Other Projects** → **DonateJython - AddGroupsUsers.py**, and right-click **Run-As** → **Administrative Script**.
2. From the resulting **Modify attributes and launch** pop-up complete the following fields:
   - **Scripting runtime**: **WebSphere Application Server v6.1**
   - **WebSphere Profile**: **ExperienceJ2EE**
   - Under **Security**:
     - Select **Specify**.
     - Enter the WebSphere administrative user ID and password defined in section 3.5 “Create the ExperienceJ2EE application server profile” on page 51.
   - Click **Apply**.
   - Click **Run**.
3. Switch to the **Console** view (in the lower-right pane), and 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.



```
<terminated> AddGroupsUsers.py [WebSphere Administrative Script] C:\ibm\appserver\java\jre\bin\java.e
WASX7209I: Connected to process "server1" on node ExperienceJ2EE
```

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,
'/opt/IBM/AppServer/optionalLibraries/jython/jython.jar'
*sys-package-mgr*: processing new jar,
'/opt/IBM/AppServer/lib/startup.jar'
...
...
```

Behind the scenes:

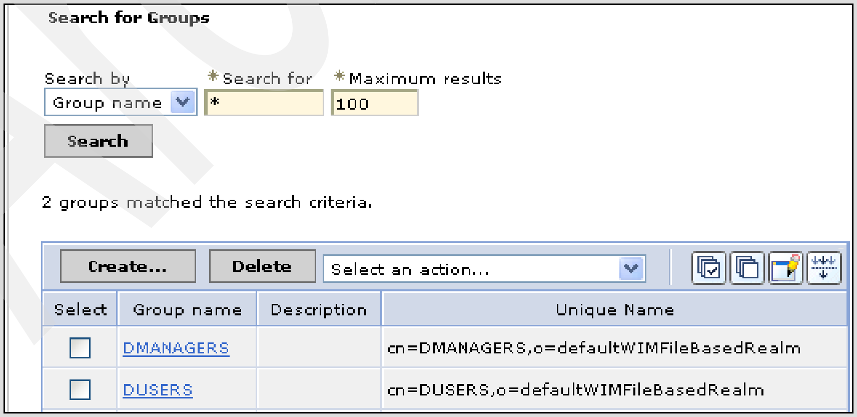
For the file-based repository, group (and user) entries are stored in fileRegistry.xml, found in the cell directory of the WAS V6.1 profile:

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 perform the following actions:

- From the **Admin Console**, on the left expand to **Users and Groups**, and click **Manage Groups**.
- In the **Manage Groups** pane, on the right click **Search**. The current groups should appear as the following illustration shows.



Search for Groups

Search by	* Search for	* Maximum results
Group name	*	100

2 groups matched the search criteria.

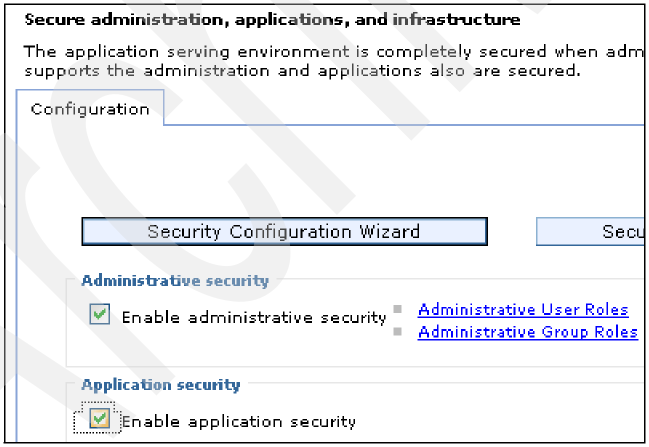
Select	Group name	Description	Unique Name
□	DMANAGERS		cn=DMANAGERS,o=defaultWIMFileBasedRealm
□	DUSERS		cn=DUSERS,o=defaultWIMFileBasedRealm

11.3 Enable J2EE application security

In this section you enable the J2EE application security. Note that you only have to enable the J2EE application security since ExperienceJ2EE Server @ localhost already has a configured user registry and already has J2EE administrative security enabled.

11.3.1 Enable J2EE application security

1. Start the **Admin Console** using the same steps as described in section “5.4.1 Start the admin console” on page 102.
   - Recall that at the **Admin Console** login screen you use the **j2eeadmin** user ID and password that you defined in section 3.5 “Create the ExperienceJ2EE application server profile” on page 51.
2. On the left expand to **Security**, and click **Secure administration, applications, and infrastructure**.
3. On the right, in **Secure administration, applications, and infrastructure**, perform the following tasks:
   - Under **Application security**, select **Enable application security**.
   - Click **Apply**.



Secure administration, applications, and infrastructure

The application serving environment is completely secured when adm
supports the administration and applications also are secured.

Configuration
Security Configuration Wizard
Secu
Administrative security
Administrative User Roles
Administrative Group Roles
Application security

☑ Enable administrative security

☑ Enable application security

4. At the top of the panel, in the **Messages** box that states “**Changes have been made to your local configuration….**”, click **Save**.
5. Logout, and close the **Admin Console**.

11.3.2 Restart the ExperienceJ2EE Server @ localhost test server

1. Return to AST V6.1, switch to the **Servers** view in the lower-right pane, select **ExperienceJ2EE Server @ localhost**, and select the re-start icon [ ] in the action bar. 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
   ```

11.4 Implement declarative security

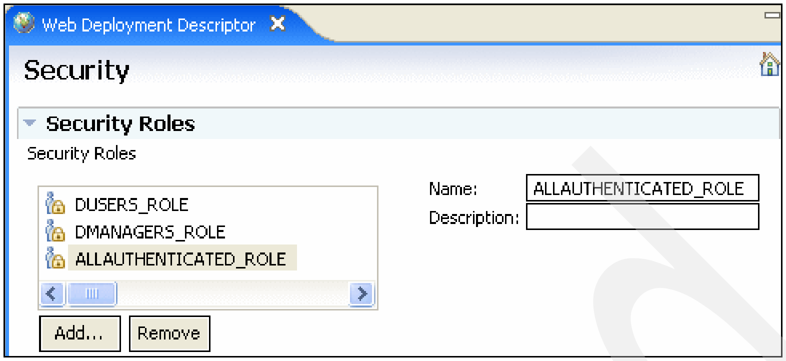
Declarative security uses deployment descriptor definitions to restrict access to either URLs (servlets, JSPs, or even HTTP files or images served by the Web container) or EJBs.

11.4.1 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 9, “Create the Web front end” on page 213.

1. Open the Web deployment descriptor, and switch to the Security settings:
   - From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **DonateWeb**, and double-click to open.
   - At the resulting **Web Deployment Descriptor** editor, switch to the **Security** tab.
2. Define the security roles by performing the following tasks:
   - Expand the **Security Roles** section, and under **Security Roles** click **Add**…
     - At the pop-up, set the name to **DUSERS_ROLE**, and click **Finish**.
   - Under **Security Roles**, click **Add…** again.
     - At the pop-up set the name to **DMANAGERS_ROLE**, and click **Finish**.
   - Under **Security Roles**, click **Add…** again.
     - At the pop-up set the name to **ALLAUTHENTICATED_ROLE**, and click **Finish**.

The Security Rules section should appear similar to the following:



Security Roles

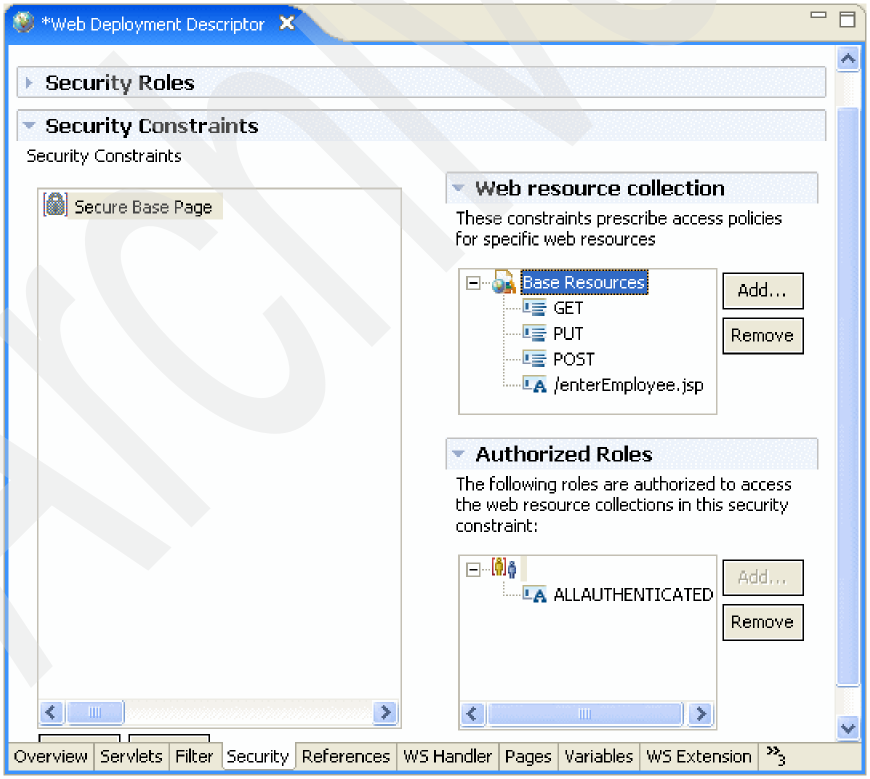
- DUSERS_ROLE
- DMANAGERS_ROLE
- ALLAUTHENTICATED_ROLE

Name: ALLAUTHENTICATED_ROLE

Description:

3. Collapse **Security Roles** and expand **Security Constraints**. From this screen you perform three separate steps: add the constraint, add the Web resource collections, and add the authorization roles.

   The following illustration shows what the Security tab looks like AFTER you add the security constraint for the enterEmployee input page.



Security Constraints

Secure Base Page

These constraints prescribe access policies for specific web resources

- Base Resources
  - GET
  - PUT
  - POST
  - /enterEmployee.jsp

The following roles are authorized to access the web resource collections in this security constraint:

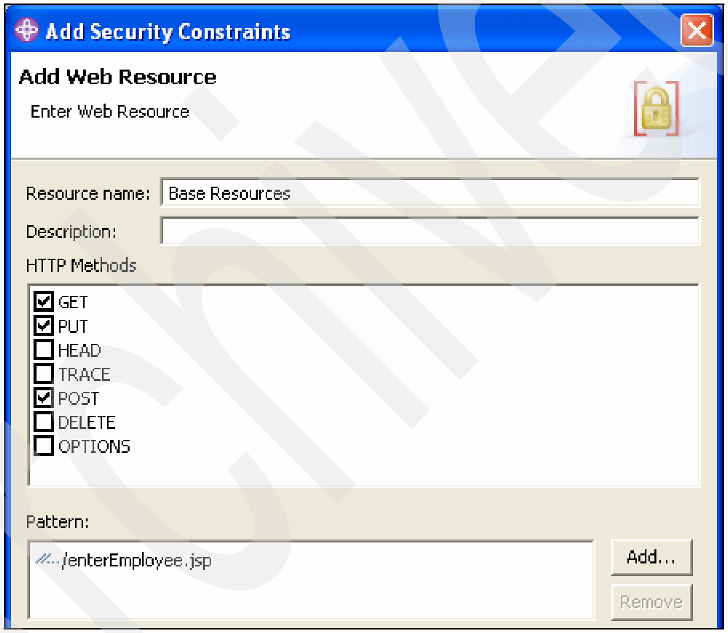
- ALLAUTHENTICATED

4. Add the security constraint to limit access to the enterEmployee input page. At the bottom left of the screen, under the **Security Constraints** section, click **Add…**.

   – (STEP 1) At the **Add Constraints** pop-up, set the name to **Secure Base Page**, and click **Next**.

   – (STEP 2) At the **Add Web Resource** pop-up complete the following fields:

   - **Resource Name**: **Base Resources**
   - **HTTP Methods**: Select **GET, PUT, POST**
   - **URL Patterns**: Click **Add**:
     - At the **Pattern** pop-up enter **/enterEmployee.jsp**, and click **OK**.
   - Click **Finish**.



Add Security Constraints

Add Web Resource

Enter Web Resource

Resource name: Base Resources

Description:

HTTP Methods

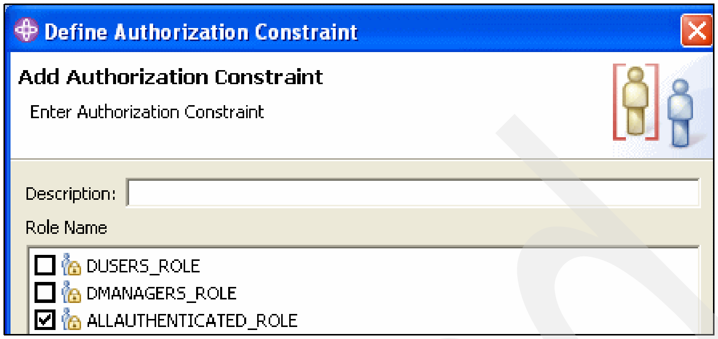
☑ GET
☑ PUT
□ HEAD
□ TRACE
☑ POST
□ DELETE
□ OPTIONS

Pattern:

/enterEmployee.jsp

   – (STEP 3) At the **Security** tab, select the new **Secure Base Page** on the left, under **Security Constraints**. On the right next to **Authorized Roles**, click **Add…**

- At the **Define Authorization Constraint** pop-up select **ALLAUTHENTICATED_ROLE**, and click **Finish**.



Define Authorization Constraint

Add Authorization Constraint

Enter Authorization Constraint

Description:

Role Name

☐ DUSERS_ROLE
☐ DMANAGERS_ROLE
☑ ALLAUTHENTICATED_ROLE

5. Add the security constraint to limit access to the employeeDetail and donateResult pages. At the bottom left of the **Security** tab (under the **Security Constraints** section), click **Add…**

   – (STEP 1) At the **Add Constraints** pop-up set the name to **Secure Result Pages**, and click **Next**.

   – (STEP 2) On the **Add Web Resource** pop-up complete the following fields:

     - **Resource Name**: **Result Resources**
     - **HTTP Methods**: Select **GET, PUT, POST**
     - **URL Patterns**: Click **Add**:
       - At the pop-up enter **/employeeDetail.jsp**, and click **OK**.
     - **URL Patterns**: Click **Add** again:
       - At the pop-up enter **/donateResult.jsp**, and click **OK**.
     - Click **Finish**.

   – (STEP 3) At the **Security** tab, select the new **Secure Result Pages** on the left (under **Security Constraints**). On the right next to **Authorized Roles**, click **Add…**

     - At the pop-up select **DUSERS_ROLE** and **DMANAGERS_ROLE**, and click **Finish**.

6. Save and close the **Web Deployment Descriptor**.

Behind the scenes*:*

These security definitions are stored in the Web deployment descriptor (actual file artifact is **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **WEB-INF** → **web.xml**), which you can view in one of three ways:

- Graphically by expanding the **Dynamic Web Projects** → **DonateWeb** → **DonateWeb** subtree and viewing the Security branch.
- Formatted View by selecting **Dynamic Web Projects** → **DonateWeb** → **DonateWeb** and double-clicking to open in the Deployment Descriptor editor and switching to the **Security** tab.
- Source view by selecting **Dynamic Web Projects** → **DonateWeb** → **DonateWeb** and double-clicking to open in the Deployment Descriptor editor and switching to the **Source** tab.

Following is the source view of the Secure Base Page constraints you just added:

```
<security-constraint>
   <display-name>
   Secure Base Page</display-name>
   <web-resource-collection>
      <web-resource-name>Base Resources</web-resource-name>
      <url-pattern>/enterEmployee.jsp</url-pattern>
      <http-method>GET</http-method>
      <http-method>PUT</http-method>
      <http-method>POST</http-method>
   </web-resource-collection>
   <auth-constraint>
      <description>
      </description>
      <role-name>ALLAUTHENTICATED_ROLE</role-name>
   </auth-constraint>
</security-constraint>
```

11.4.2 Restrict access to Donate session EJB via declarative security

In this section you implement EJB declarative security to restrict access to the Donate session EJB created in Chapter 7, “Create the donate business logic element” on page 165.

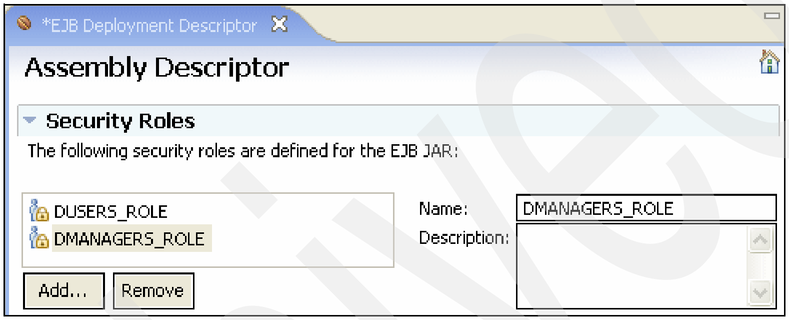
1. Open the EJB deployment descriptor, and switch to the Assembly settings.
   – From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2**, and double-click to open.

- At the resulting **EJB Deployment Descriptor** editor, switch to the **Assembly** tab.

2. Define the security roles.

   - From the upper left ensure that the **Security Roles** section is expanded and, under **Security Roles** click **Add…**.
     - At the pop-up set the name to **DUSERS_ROLE**, and click **Finish**.
   - Under **Security Roles** click **Add…** again:
     - At the pop-up set the name to **DMANAGERS_ROLE**, and click **Finish**.

   The Security Rules section should appear similar to the following:



*EJB Deployment Descriptor

Assembly Descriptor

Security Roles

The following security roles are defined for the EJB JAR:

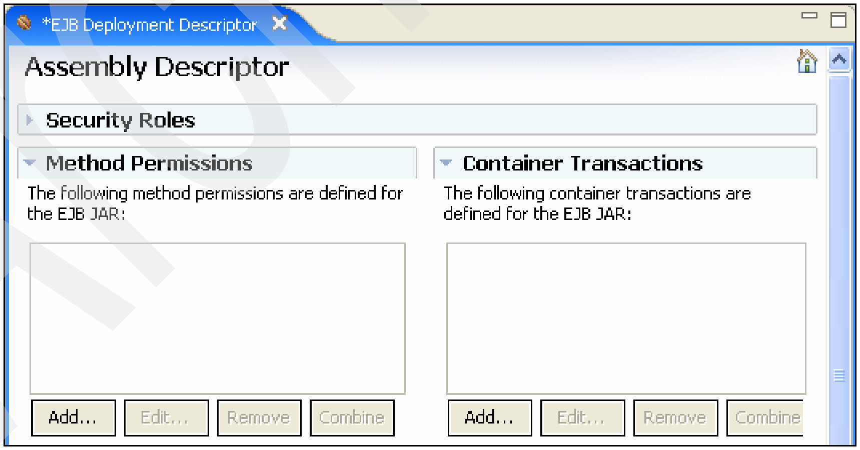
- DUSERS_ROLE
- DMANAGERS_ROLE

Name: DMANAGERS_ROLE

Description:

3. Define the Method Permissions using the following instructions:

   - In **the EJB Deployment Descriptor** → **Assembly** tab, ensure that **Method Permissions** is expanded. Under it select **Add…**



*EJB Deployment Descriptor

Assembly Descriptor

Security Roles

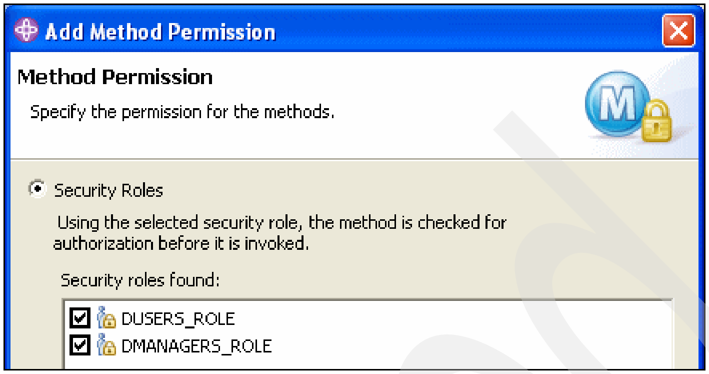
Method Permissions

The following method permissions are defined for the EJB JAR:

Container Transactions

The following container transactions are defined for the EJB JAR:

- At the **Method Permissions** pop-up, select **Security Roles**, select both **DUSERS _ROLE** and **DMANAGERS_ROLE**, and click **Next**.



Add Method Permission

Method Permission

Specify the permission for the methods.

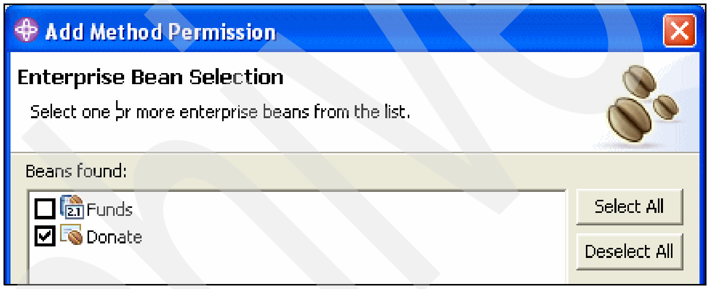
● Security Roles

Using the selected security role, the method is checked for authorization before it is invoked.

Security roles found:

☑ DUSERS_ROLE
☑ DMANAGERS_ROLE

- At the **Enterprise Bean Selection** pop-up, select **Donate**, and click **Next**.



Add Method Permission

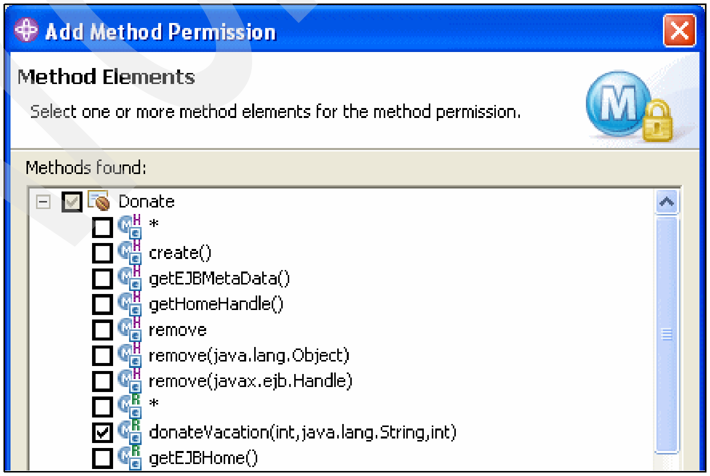
Enterprise Bean Selection

Select one or more enterprise beans from the list.

Beans found:

☐ Funds
☑ Donate

- At the **Method Elements** pop-up, expand **Donate**, select **donateVacation(…)**, and click **Finish**.



Add Method Permission

Method Elements

Select one or more method elements for the method permission.

Methods found:

⊟ ☑ Donate
☐ *
☐ create()
☐ getEJBMetaData()
☐ getHomeHandle()
☐ remove
☐ remove(java.lang.Object)
☐ remove(javax.ejb.Handle)
☐ *
☑ donateVacation(int,java.lang.String,int)
☐ getEJBHome()

11.4.3 Update the EJB method permissions to apply to all interfaces

Starting in J2EE 1.4, EJB method permissions can be defined separately for each specific interface—Remote, Home, Local, LocalHome, and ServiceEndpoint or a default permission for all interfaces (Unspecified). This allows the developer to set the following types of security constraints:

- To apply one set of security constraints for one interface, such as for the Local, which can only be accessed by other “more trusted” EJBs running on the application server.
- Another set of security constraints for another interface, such as for ServiceEndpoint, which handles Web service requests that could be generated by external and presumably “less trusted” entities.

The wizard, for creating the EJB method permission, creates the permission for a single interface (Remote). If you left this default value, this permission does not apply to the Web service, or ServiceEndpoint, that you will create and secure in the following chapters:

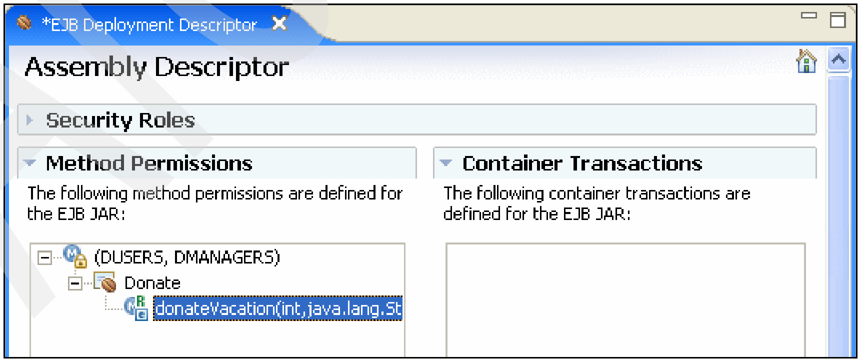
- Create in “Chapter 13. Create the Web service” on page 305.
- Secure in “Chapter 14. Implement security for the Web service” on page 331.

You have two options to secure the EJB Web service interface:

- Change the method permission created here to apply to all interfaces.
- Create a second method permission after you create the Web service.

In this scenario, you change the method permission to apply to all interfaces.

1. Still in the **EJB Deployment Descriptor**, on the **Assembly Descriptor** tab, expand the selections under **Method Permissions**, and select **donateVacation(int, java.lang.String, int)**.



The following method permissions are defined for the EJB JAR:

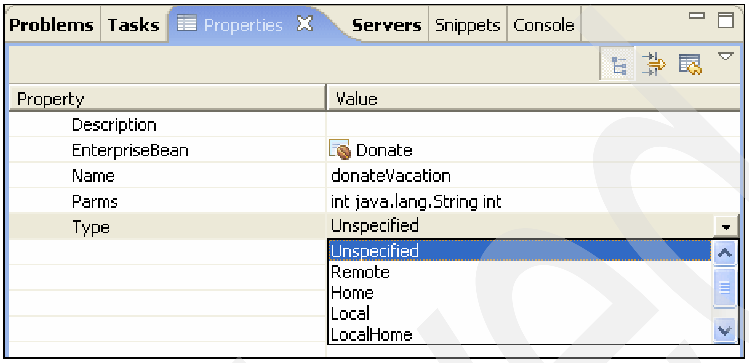
(DUSERS, DMANAGERS)
Donate
donateVacation(int,java.lang.St

The following container transactions are defined for the EJB JAR:

2. In the lower-right pane, switch to the **Properties** view. If the **Properties** view is not visible, from the action bar run **Window** → **Show View** → **Other…**, and

from the resulting **Show View** pop-up select **Basic** → **Properties**, and click **OK**.

– Change the value for property **Type** from **Remote** to **Unspecified** using the pull-down on the right of the Value column. The pull-down is not visible until you select the field.



Property	Value
Description	
EnterpriseBean	Donate
Name	donateVacation
Parms	int java.lang.String int
Type	Unspecified

3. Save and close the **EJB Deployment Descriptor**.

Behind the scenes:

These security definitions are stored in the EJB deployment descriptor. The actual file artifact is **EJB Projects** → **DonateEJB2** → **ejbModule** → **META-INF** → **ejb-jar.xml**. Note that the section defining the method permission for donateVacation should contain the following:

```
<assembly-descriptor>
   <security-role>
      <role-name>DUSERS</role-name>
   </security-role>
   <security-role>
      <role-name>DMANAGERS</role-name>
   </security-role>
   <method-permission>
      <role-name>DUSERS</role-name>
      <role-name>DMANAGERS</role-name>
      <method>
         <ejb-name>Donate</ejb-name>
         <method-name>donateVacation</method-name>
         <method-params>
            <method-param>int</method-param>
            <method-param>java.lang.String</method-param>
            <method-param>int</method-param>
         </method-params>
      </method>
   </method-permission>
</assembly-descriptor>
```

and should NOT contain the following line (which would restrict the method permission to the remote interface only):

```
<method-intf>Remote</method-intf>
```

11.4.4 Update the EAR by gathering roles and assigning users

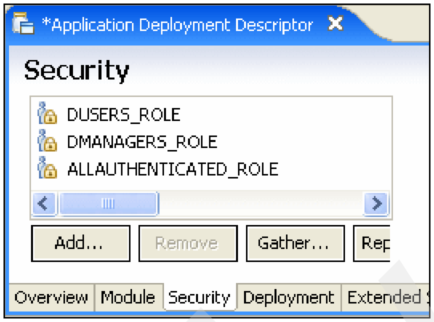
The EAR deployment descriptor contains the mapping of role information to specific users or groups. Therefore, you first discover, or gather, the roles from the included Web and EJB projects, and then define the mapping.

1. Open the deployment descriptor and switch to the Security settings.
   - From the **Project Explorer**, select **Enterprise Applications** → **DonateEAR** → **DonateEAR**, and double-click to open.

– At the resulting **Application Deployment Descriptor** editor, switch to the **Security** tab.

2. Gather the roles from the various sub-modules.

   – Click **Gather…** This should import the DUSERS_ROLE, DMANAGERS_ROLE and ALLAUTHENTICATED_ROLE roles from DonateWeb and DonateEJB2.



*Application Deployment Descriptor

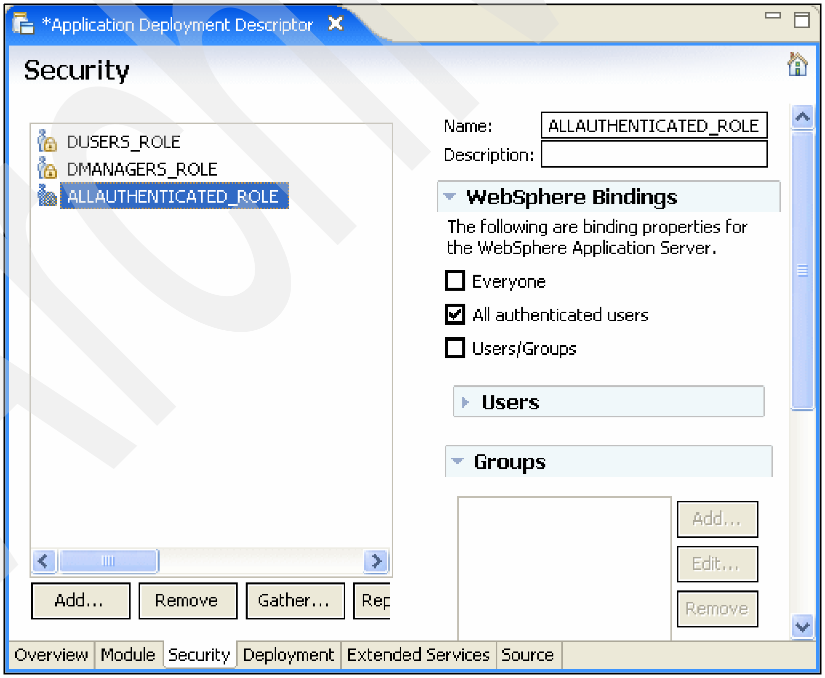
Security

- DUSERS_ROLE
- DMANAGERS_ROLE
- ALLAUTHENTICATED_ROLE

3. Assign the users to ALLAUTHENTICATED_ROLE.

   – Under **Security**, select **ALLAUTHENTICATED_ROLE**.

   – Under **WebSphere Bindings**, select **All authenticated users**.



*Application Deployment Descriptor

Security

- DUSERS_ROLE
- DMANAGERS_ROLE
- ALLAUTHENTICATED_ROLE

Name: ALLAUTHENTICATED_ROLE

Description:

WebSphere Bindings

The following are binding properties for the WebSphere Application Server.

☐ Everyone

☑ All authenticated users

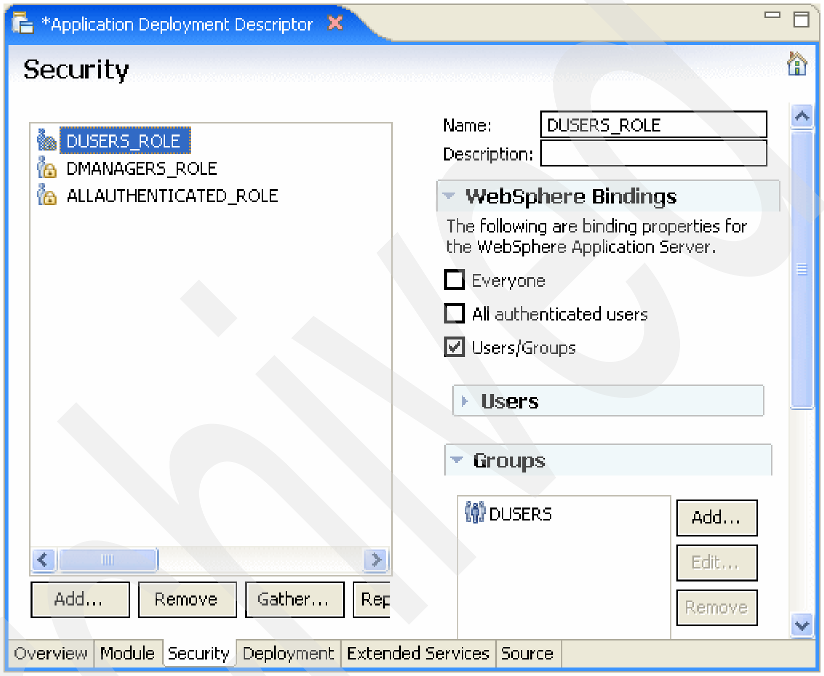
☐ Users/Groups

Users

Groups

4. Assign the group to DUSERS_ROLE by performing the following tasks:

- Under **Security**, select **DUSERS_ROLE**.
- Under **WebSphere Bindings**, select **Users/Groups**.
- Under **WebSphere Bindings**, expand **Groups**, and then select **Add…** Ensure that you are under Groups and not under Users.
  - Group Name: **DUSERS**
- Click **Finish**.



Name: DUSERS_ROLE

Description:

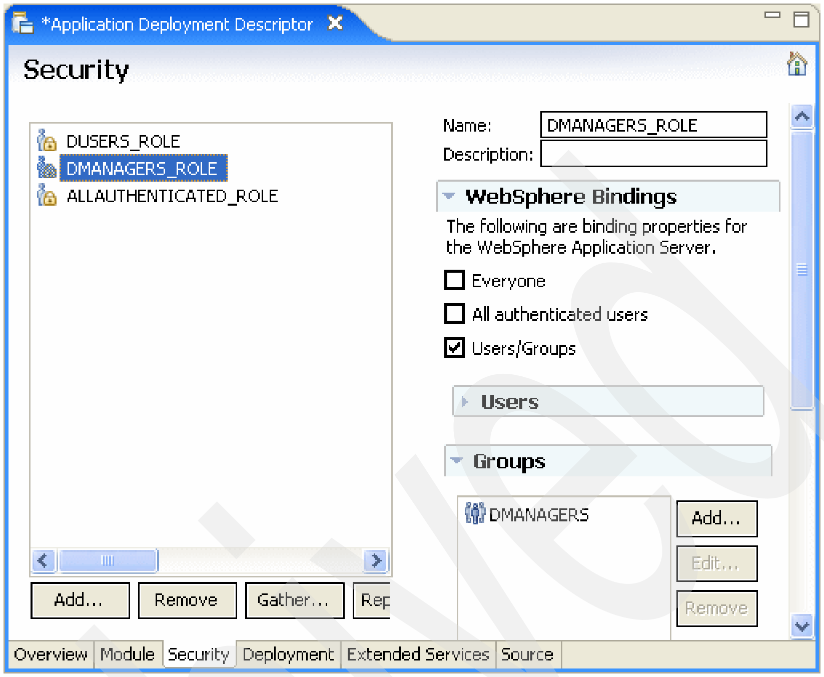
□ Everyone

□ All authenticated users

☑ Users/Groups

DUSERS

5. Assign the group to DMANAGERS_ROLE by performing the following tasks:
   - Under **Security**, select **DMANAGERS_ROLE**.
   - Under **WebSphere Bindings**, select **Users/Groups**.
   - Under **WebSphere Bindings**, expand **Groups**, and then select **Add…**. Ensure that you are under Groups and not under Users.
     - **Group Name**: **DMANAGERS**
   - Click **Finish**.



Name: DMANAGERS_ROLE

Description:

The following are binding properties for the WebSphere Application Server.

□ Everyone

□ All authenticated users

☑ Users/Groups

6. Save and close the **Application Deployment Descriptor**.

Behind the scenes:

The gather operation examines all included projects (EJB, Web, and Application Client) and defines the roles in the EAR deployment descriptor (**Enterprise Applications** → **DonateEAR** → **EarContent** → **META-INF** → **application.xml**):

```
<security-role id="SecurityRole_1152292115671">
   <role-name>DUSERS_ROLE</role-name>
</security-role>
<security-role id="SecurityRole_1152292115672">
   <role-name>DMANAGERS_ROLE</role-name>
</security-role>
<security-role id="SecurityRole_1152292115609">
   <role-name>ALLAUTHENTICATED_ROLE</role-name>
</security-role>
```

The mapping of these roles to users, groups, and special definitions is contained in the IBM specific binding file (**Enterprise Applications** → **DonateEAR** → **EarContent** → **META-INF** → **ibm-application-bnd.xmi)**:

```
<authorizations xmi:id="RoleAssignment_1152292115593">
   <specialSubjects
      xmi:type="applicationbnd:AllAuthenticatedUsers"
      xmi:id="AllAuthenticatedUsers_1152292115609"
      name="AllAuthenticatedUsers"/>
   <role
      href="META-INF/application.xml#SecurityRole_1152292115609"/>
</authorizations>
<authorizations xmi:id="RoleAssignment_1152292115671">
   <role
      href="META-INF/application.xml#SecurityRole_1152292115671"/>
    <groups xmi:id="Group_1152292353531" name="DUSERS"/>
</authorizations>
<authorizations xmi:id="RoleAssignment_1152292115672">
   <role
      href="META-INF/application.xml#SecurityRole_1152292115672"/>
   <groups xmi:id="Group_1152292115672" name="DMANAGERS"/>
</authorizations>
```

Why an IBM specific binding file?

The J2EE specification is very extensive, but is by no means complete and defers the specification of some areas to the J2EE runtime providers, such as the mapping of security roles to users and groups.

The following excerpt is from **Designing Enterprise Applications with the J2EE Platform, Second Edition**, which you can find at the following Web address:

`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/deployment/deployment6.html`

> ***7.5.2.1 Vendor-Specific Deployment Information***
>
> *The J2EE platform specification defines deployment unit requirements for each of the four J2EE module types and for a J2EE application itself. Each specification defines how the archive file must be structured to operate correctly with a J2EE deployment tool. In addition to application code and a deployment descriptor, an application requires a certain amount of additional vendor- or environment-specific binding information. For example, when the J2EE reference implementation receives an application EAR file from a deployer, it also needs the following information:*
>
> - *A JNDI name for each enterprise bean's home interface*
> - *A mapping of the application's abstract security roles to user and group names*
> - *JNDI lookup names and account information for all databases*
> - *JavaMail™ session configuration information*

As a result, the implementation differs between J2EE providers, and the J2EE application must be updated when deploying on different J2EE application servers. For example, the BEA WebLogic server puts the binding information in weblogic.xml, weblogic-ejb.xml, or weblogic-application.xml, and the Sun Java System Application Server uses a similar convention.

11.4.5 Test declarative security

In this section you verify that the security mechanisms operate correctly. Table 11-1 summarizes the assignments you made in the previous sections and links the J2EE roles to the file registry groups to the file registry users.

Table 11-1 Summary of previous assignments

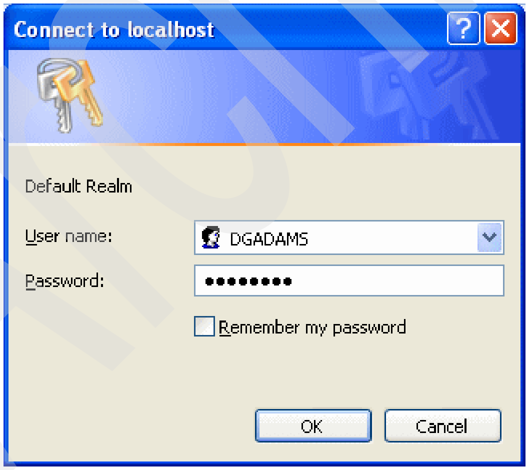
J2EE role	Group	User members
DUSERS_ROLE	DUSERS	DCBROWN, DNARMSTRONG
DMANAGERS_ROLE	DMANAGERS	DNARMSTRONG
ALLAUTHENTICATED_ROLE	<no group>	All users (j2eeadmin, DCBROWN, DNARMSTRONG, DGADAMS)

Testing occurs through an external browser because the AST V6.1 browser keeps credentials once established, making it difficult to validate security.

1. Test the access using DGADAMS:
   – Open a browser and type one of the following Web addresses:

     ```
     http://localhost:9080/DonateWeb/enterEmployee.jsp
     https://localhost:9443/DonateWeb/enterEmployee.jsp
     ```

   – At the pop-up, login as DGADAMS using the password defined in section 11.2.1 “Create and configure the Jython project and script” on page 261.



Connect to localhost

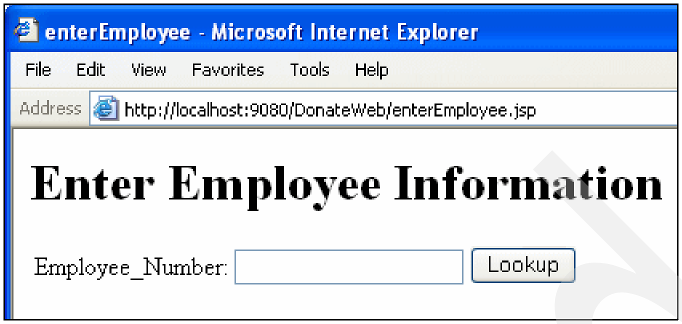
Default Realm

User name: DGADAMS

Password: •••••••••

□ Remember my password

The initial access to enterEmployee.jsp succeeds because the declarative security for this page allows access by anyone belonging to the ALLAUTHENTICATED_ROLE, and all authenticated users are assigned to that role.


enterEmployee - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:9080/DonateWeb/enterEmployee.jsp
Lookup

Enter Employee Information

Employee_Number:

Off course?

If you do not see the login pop-up perform the following troubleshooting tasks:

- Ensure that you correctly created the Web declarative security in section 11.4.1 “Restrict access to Web pages via declarative security” on page 266 for the Base pages.
- Ensure that you enabled the J2EE application security in section 11.3.1 “Enable J2EE application security” on page 265.

Behind the scenes:

The previous logon prompt is the BASIC authentication mechanism and is the default option for requesting authentication information from a browser-based client. The authentication information 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: Utilizes digital certificates passed over an SSL connection.
- Digest: Similar to BASIC, but the password is transmitted using a custom encryption mechanism.

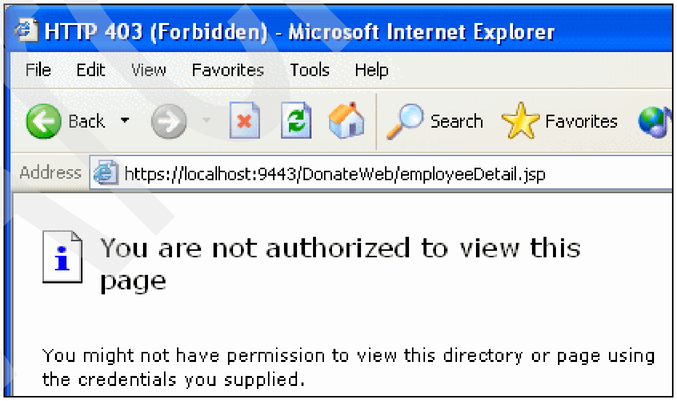
These options are set on the Page tab of the Web Deployment Descriptor editor for the encompassing Web Application.

Additional information is available in the SUN J2EE tutorial in the section "**Understanding Login Authentication**" at the following Web site:

`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Security5.html`

2. Try to retrieve an employee record (using DGADAMS) using the following instructions:

   – Enter a valid employee number (such as 1) in the **Employee_Number** field, and click **Lookup**. The following failure message appears in the browser:



https://localhost:9443/DonateWeb/employeeDetail.jsp

You are not authorized to view this page

You might not have permission to view this directory or page using the credentials you supplied.

The access to the secondary elements, in this case the employeeDetail.jsp page, should fail 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. The following message may also appear in the **Console** view:

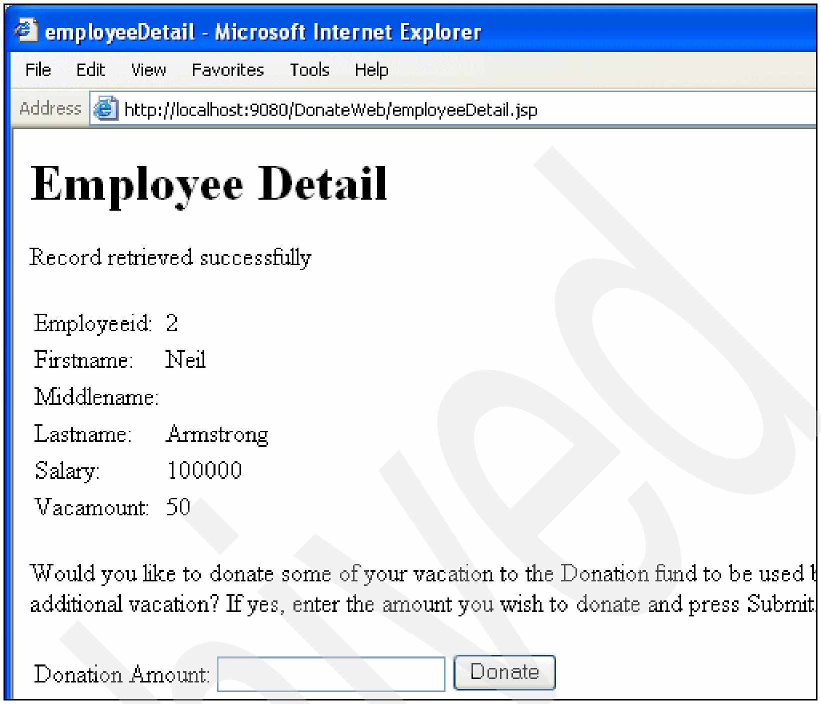
```
SECJ0129E: Authorization failed for DGADAMS while invoking
POST on default_host:DonateWeb/employeeDetail.jsp,
Authorization failed, Not granted any of the required roles:
DUSERS DMANAGERS
```

> **Off course**?
>
> If this access works, and you in fact see the employeeDetail.jsp page, then ensure that you correctly created the Web declarative security in section 11.4.1 “Restrict access to Web pages via declarative security” on page 266 for the Result pages.

3. Test the access using DNARMSTRONG:
   - Close the browser.
   - Open a new browser and type the following Web address:

     `http://localhost:9080/DonateWeb/enterEmployee.jsp`
   - Login as DNARMSTRONG.

     As previously, the initial access to enterEmployee.jsp succeeds because the declarative security for this page allows access by anyone belonging to the ALLAUTHENTICATED role.
   - Enter an employee number of 2, and click **Lookup**. The access to the secondary element (employeeDetail.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 group that is assigned to the DMANAGERS_ ROLE role.



employeeDetail - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:9080/DonateWeb/employeeDetail.jsp
Donate

Employee Detail

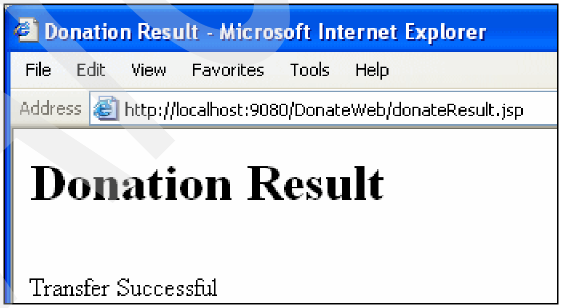
Record retrieved successfully

Employeeid: 2
Firstname: Neil
Middlename:
Lastname: Armstrong
Salary: 100000
Vacamount: 50

Would you like to donate some of your vacation to the Donation fund to be used b
additional vacation? If yes, enter the amount you wish to donate and press Submit

Donation Amount:

- From the same browser, type a donation amount, and click **Donate**. The access to the secondary elements (the Donate session EJB and donateResult.jsp) succeeded 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.



Donation Result - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:9080/DonateWeb/donateResult.jsp

Donation Result

Transfer Successful

Off course?

If you receive an error on the donation action, ensure that you correctly created the EJB declarative security in section 11.4.2 “Restrict access to Donate session EJB via declarative security” on page 270.

Extra credit:

Optionally try the above steps using DCBROWN. The access should succeed because DCBROWN belongs to the DUSERS_ROLE role.

11.5 Implement programmatic security

Programmatic security is implemented in user code It references the role definitions in the deployment descriptors.

11.5.1 Implement programmatic security in EmployeeFacade

In this section you update the EmployeeFacade session EJB to restrict access to the salary field only to members of the DMANAGERS_ROLE role.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB** → **DonateEJB** → **Session Beans** → **EmployeeFacade** → **EmployeeFacadeBean**(.java), and double-click to open.

   – Locate the method **getEmployeeByKey(EmployeeKey primaryKey)**.

   – Locate the following line:

   ```
   emp0.setSalary(anEmployeeLocal.getSalary());
   ```

   Insert the following line before it:

   ```
   if (mySessionCtx.isCallerInRole("DMANAGERS_REF"))
   ```

Behind the scenes:

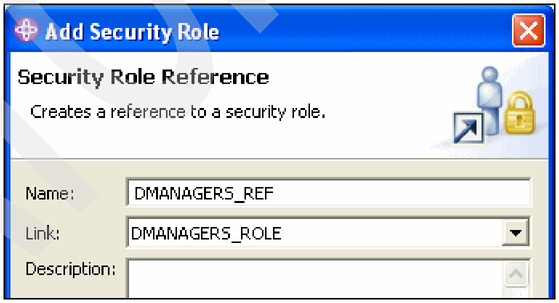
This salary field is only copied to the Employee object (emp0) when the user belongs to the role pointed to by the DMANAGERS_REF role reference.

You configure this role reference in the next step.

- Save and close **EmployeeFacadeBean.java**.

2. Update the EJB deployment descriptor to include a reference to the DMANAGERS_ROLE role utilized in the EmployeeFacadeBean code.

   - From the **Project Explorer**, select **EJB Projects** → **DonateEJB** → **DonateEJB**, and double-click to open.
   - At the resulting **EJB Deployment Descriptor** editor, switch to the **Assembly** tab, and perform the following tasks:
     - Ensure that the **Security Roles** section in the upper left is expanded, and then click **Add…**
     - At the pop-up, set the name to **DMANAGERS_ROLE**.
     - Click **Finish**.
   - Switch to the **References** tab, and perform the following tasks:
     - Under **References**, select **EmployeeFacade**, and click **Add…**
     - At the **Add Reference** pop-up, select **Security role reference**, and click **Next**.
     - At the **Add Security Role** pop-up, perform the following tasks:
       - Under **Link**, select **DMANAGERS_ROLE** from the pull-down.
       - Under **Name**, enter **DMANAGERS_REF**. Note that you must change this value from the default (generated when you set the Link value), because DMANAGERS_REF is the name that you used previously when updating the getEmployeeByKey method.
       - Click **Finish**.



Security Role Reference

Creates a reference to a security role.

Name: DMANAGERS_REF

Link: DMANAGERS_ROLE

Description:

   - Save and close the **EJB Deployment Descriptor**

Behind the scenes:

These security definitions are stored in the EJB deployment descriptor (actual file artifact is **EJB Projects** → **DonateEJB** → **ejbModule** → **META-INF** → **ejb-jar.xml**).

Adding the DMANAGERS_ROLE security role (under the Assembly tab) defined the role:

```
<assembly-descriptor>
   <security-role>
      <role-name>DMANAGERS_ROLE</role-name>
   </security-role>
</assembly-descriptor>
```

Adding the DMANAGERS_REF Reference (under the Reference tab) provides the linkage between this reference (used in the updated EmployeeFacadeBean.java) and the actual DMANAGERS:

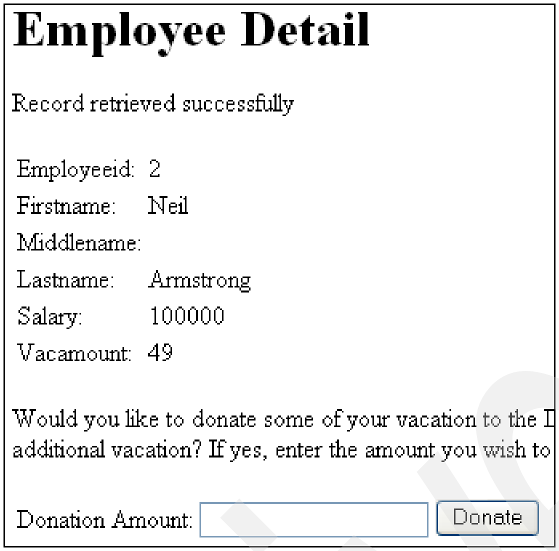
```
<security-role-ref>
   <description>
   </description>
   <role-name>DMANAGERS_REF</role-name>
   <role-link>DMANAGERS_ROLE</role-link>
</security-role-ref>
```

11.5.2 Test programmatic security in the EmployeeFacade session EJB

1. Test the access using DNARMSTRONG by performing the following tasks:
   - Open a new browser, and type the following Web address:

     `http://localhost:9080/DonateWeb/enterEmployee.jsp`
   - Log in as DNARMSTRONG.
   - At 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 to this information



Employee Detail

Record retrieved successfully

Employeeid:	2
Firstname:	Neil
Middlename:	
Lastname:	Armstrong
Salary:	100000
Vacamount:	49

Would you like to donate some of your vacation to the I
additional vacation? If yes, enter the amount you wish to

Donation Amount:

2. Test the access using DCBROWN by performing the following tasks:

   – Close the browser.

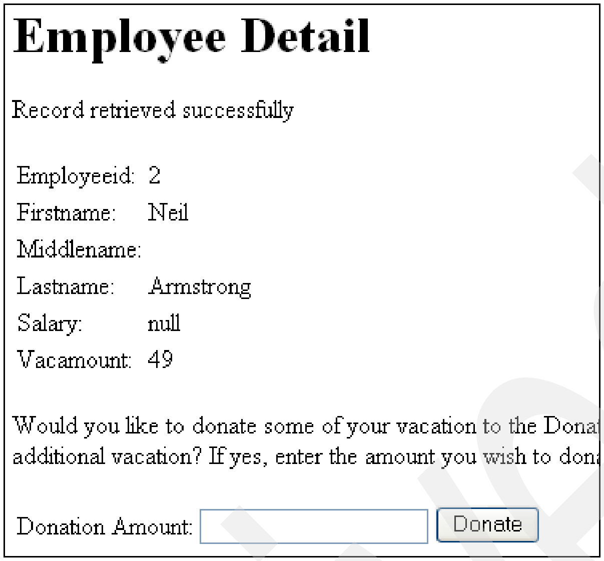
   – Open a new browser, and type the following Web address:

     `http://localhost:9080/DonateWeb/enterEmployee.jsp`

   – Log in as DCBROWN.

   – At the resulting page, retrieve record 2.

   Note that salary information is nulled out 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 Detail

Record retrieved successfully

Employeeid: 2
Firstname: Neil
Middlename:
Lastname: Armstrong
Salary: null
Vacamount: 49

Would you like to donate some of your vacation to the Donat
additional vacation? If yes, enter the amount you wish to don

Donation Amount:

3. Close the browser.

11.5.3 Implement programmatic security in DonateWeb

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.

You use programmatic security in the employeeDetail.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. From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **employeeDetail.jsp**, and double-click to open.
   - Locate the following lines, which display the table row with the salary information:

```
<tr>
   <td  width="" align="left">Salary:</td>
   <td><%= record.getEmployee_Details().getSalary() %></td>
</tr>
```

   - Before the table row lines (for example, before the opening <TR> shown above), add the following:

```
<%
if (request.isUserInRole("DMANAGERS_ROLE"))
{
%>
```

– After the table row lines (for example, after the closing </TR> shown above), add the following:

```
<%
}
%>
```

– Save and close **displayEmployee.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 display field is skipped.

When finished, this section will contain the following (the added lines are highlighted):

```
<%
if (request.isUserInRole("DMANAGERS_ROLE"))
{
%>
<tr>
   <td  width="" align="left">Salary:</td>
   <td><%= record.getEmployee_Details().getSalary() %></td>
</tr>
<%
}
%>
```

Also, note that the JSP accesses the Security role directly (and not through a reference). Why?

Answer: The J2EE Web deployment descriptor does not support security role references.

11.5.4 Test programmatic security in DonateWeb

1. Test the access using DNARMSTRONG by performing the following tasks:

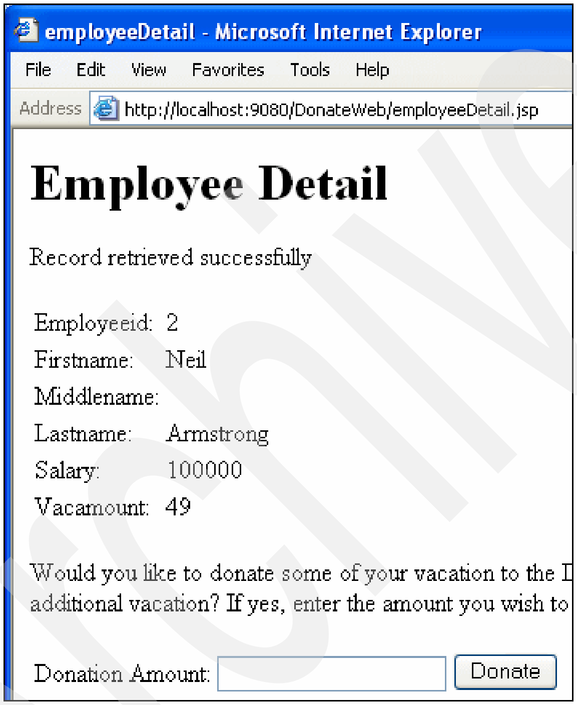
   – Open a browser, and type the following Web address:

   `http://localhost:9080/DonateWeb/enterEmployee.jsp`

   – Log in as DNARMSTRONG.

   – At the resulting page, retrieve record 2.

   Note that the salary field is shown because DNARMSTRONG belongs to the DMANAGERS group, which belongs to the DMANAGERS_ROLE role.


employeeDetail - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:9080/DonateWeb/employeeDetail.jsp

Employee Detail

Record retrieved successfully

Employeeid: 2
Firstname: Neil
Middlename:
Lastname: Armstrong
Salary: 100000
Vacamount: 49

Would you like to donate some of your vacation to the I
additional vacation? If yes, enter the amount you wish to

Donation Amount: Donate

2. Test the access using DCBROWN by performing the following tasks:

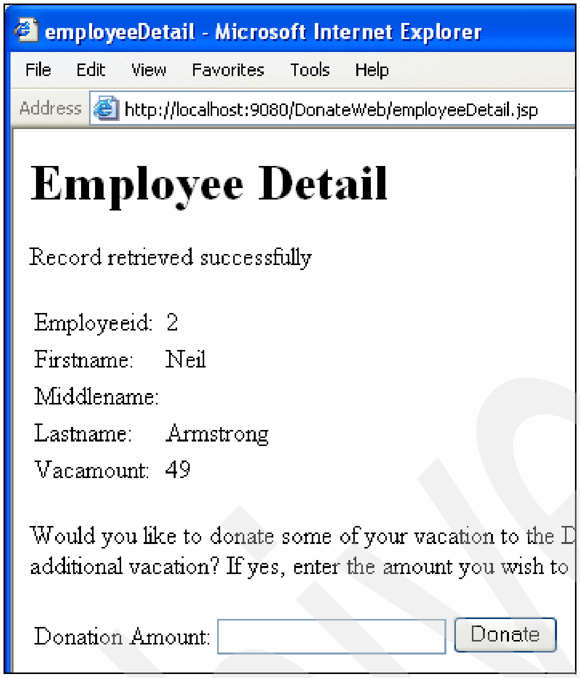
   – Close the browser, open new a browser, and type the following Web address:

   `http://localhost:9080/DonateWeb/enterEmployee.jsp`

   – Log in as DCBROWN.

   – At the resulting page, retrieve record 2.

   Note that the salary field is not displayed because DCBROWN does not belong to the DMANAGERS_ROLE role.


employeeDetail - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:9080/DonateWeb/employeeDetail.jsp
Donate

Employee Detail

Record retrieved successfully

Employeeid: 2
Firstname: Neil
Middlename:
Lastname: Armstrong
Vacamount: 49

Would you like to donate some of your vacation to the D
additional vacation? If yes, enter the amount you wish to

Donation Amount:

3. Close the browser.

11.6 Update the J2EE application client for security

The J2EE 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 10.5 “Test the application client” on page 251.
- Through a properties file, such as sas.client.props.
- Through programmatic login.

This example utilizes the properties file methods.

11.6.1 Create a customized properties file

1. Windows : From a command prompt, copy:

   ```
   C:\IBM\AppServer\profiles\ExperienceJ2EE\properties\sas.client.pr
   ops
   ```

   to:

   ```
   C:\ExperienceJ2EE\ejbclient.props
   ```

2. Linux : From a terminal session, copy:

   ```
   /opt/IBM/AppServer/profiles/ExperienceJ2EE/properties/sas.client.
   props
   ```

   to:

   ```
   $HOME/ExperienceJ2EE/ejbclient.props
   ```

   where $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. 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**

4. Save and close.

Extra credit:

The WAS V6.1 **PropFilePasswordEncoder** utility encrypts the passwords in property files.

For example, if you ran the utility under windows:

```
C:
CD \ExperienceJ2EE>\
C:\IBM\AppServer\profiles\ExperienceJ2EE\bin\PropFilePasswordEn
coder.bat ejbclient.props -SAS
```

The com.ibm.CORBA.loginPassword keyword value is encrypted:

```
com.ibm.CORBA.loginPassword={xor}JycnJycnJyc\=
```

11.6.2 Update and test the J2EE application client test configuration

1. From the **J2EE Perspective** action bar, select **Run** → **Run**…

   – Under **Configurations**, select the **WebSphere v6.1 Application Client** → **DonateClient** launch configuration.

   – Windows: Switch to the **Arguments** tab, and change this value under **VM arguments** from:

   ```
   -Dcom.ibm.CORBA.ConfigURL=file:C:/IBM/AppServer/profiles/AppSr
   v01/properties/sas.client.props
   ```

   to

   ```
   -Dcom.ibm.CORBA.ConfigURL=file:C:/ExperienceJ2EE/ejbclient.pro
   ps
   ```

   – Linux: Switch to the **Arguments** tab, and change this value under **VM arguments** from:

   ```
   -Dcom.ibm.CORBA.ConfigURL=file:/opt/IBM/AppServer/profiles/App
   Srv01/properties/sas.client.props
   ```

   to

   ```
   -Dcom.ibm.CORBA.ConfigURL=file:/home/<user>/ExperienceJ2EE/ejb
   client.props
   ```

   where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.

   – Click **Apply**, and then click **Run**.

   – 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.

> **Off course**?
>
> If you do not see the DonateClient output in the **Console** view, select the Donate Client output from the Display Selected Console icon pull-down [🖥 ▾].
>
> The **Console** view automatically switches to the most recent output, so it is very possible that a message on the ExperienceJ2EE Server @ localhost could cause a switch.

11.7 Disable J2EE application security

The next scenario in this book, which is in Chapter 13, “Create the Web service” on page 305, requires that the test server run without J2EE application security. Therefore, in this section you disable the J2EE application security that you enabled in section 11.3 “Enable J2EE application security” on page 265.

11.7.1 Disable J2EE application security

1. Start the **Admin Console** using the steps provided in section “5.4.1 Start the admin console” on page 102.
   - Recall that at the **Admin Console** login screen you use the **j2eeadmin** user ID and password that you defined in section 3.5 “Create the ExperienceJ2EE application server profile” on page 51.
2. On the left, expand to **Security**, and click **Secure administration, applications, and infrastructure**.
3. On the right, in **Secure administration, applications, and infrastructure**, perform the following tasks:
   - Under **Application security**, clear **Enable application security**.
   - Click **Apply**.
4. Save the changes by going to the top of the panel, in the **Messages** box that states “**Changes have been made to your local configuration….**”, and click **Save**.
5. Logout and close the **Admin Console**.

11.7.2 Restart the ExperienceJ2EE Server @ localhost test server

1. In AST V6.1, switch to the **Servers** view in the lower-right pane, select **ExperienceJ2EE Server @ localhost**, and select the re-start icon [ ] in the action bar. 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
```

11.8 Explore!

J2EE security contains many aspects, and addressing all of them would require a separate redbook (and in fact that was done -- see the reference list in the learn section). Some of the other security aspects include:

- **Single sign-on**: The J2EE user registry by default provides the authenticated user with a Lightweight Third Party Authentication (LTPA) token that is then used when the user accesses J2EE artifacts to determine the validated username for the role-based access.

  For this book, the LTPA configuration is valid only within the single-server instance (of ExperienceJ2EE Server @ localhost). However, you can enable this token to work with other WebSphere Application Servers (or with any other application that supports LTPA) by configuring them to work with the same key set. With a shared LTPA key set, one application recognizes and accepts the LTPA token generated by another.

  Earlier versions of Application Server also supported a single-application token (only valid in the context of a single J2EE application) called Simple WebSphere Authentication Mechanism (SWAM). This mechanism is deprecated in WAS V6.1.

- **Java 2 Security**: This is a 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. Once Java 2 Security is enabled, by default all access is denied unless explicitly granted by a policy file. This is the opposite of J2EE application role based security, where by default access is allowed unless explicitly denied.

- **Java Authorization Contract for Containers (JACC)**: Provides a common interface for connecting J2EE application servers with an external authorization provider (such as IBM Tivoli® Access Manager). This allows the J2EE application server authorization definitions to be managed by a centralized authorization system instead of by an internal J2EE unique mechanism.

  JACC requires that J2EE containers be configurable to point to an external authorization provider, both for propagating policy information when installing and removing a J2EE application and when making runtime authorization decisions.

Additional learn resources are available at the following Web addresses:

- WAS V6.1 InfoCenter: Implementing single sign-on to minimize Web user authentications:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/tsec_msso.html

- WAS V6.1 InfoCenter: Java 2 Security:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/csec_rsecmgr2.html

- WAS V6.1 InfoCenter: JACC support in WebSphere Application Server:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.base.doc/info/aes/ae/csec_jaccsupport.html

12

Alternative Import the core J2EE application

This chapter describe an alternative to implementing the Donate enterprise application as described in the preceding chapters in Part 2, “Core J2EE application” .

This alternative is intended for those who are familiar with the core J2EE constructs and capabilities with AST V6.1 and WAS V6.1 and are primarily interested in utilizing the Web services and JMS capabilities described in subsequent sections.

12.1 Preliminary activities

Complete the preliminary activities described in Part 1, “Preliminary activities” :

1. To install the required software and fixes, follow the instructions in “Chapter 2. Install and configure software” on page 23.
2. To create and configure the AST V6.1 workspace, follow the instructions in “Chapter 3. Configure the development environment” on page 41.
3. To create and configure the required Derby database, follow the instructions “Chapter 4. Prepare the legacy application” on page 71.

12.2 Import the sample application

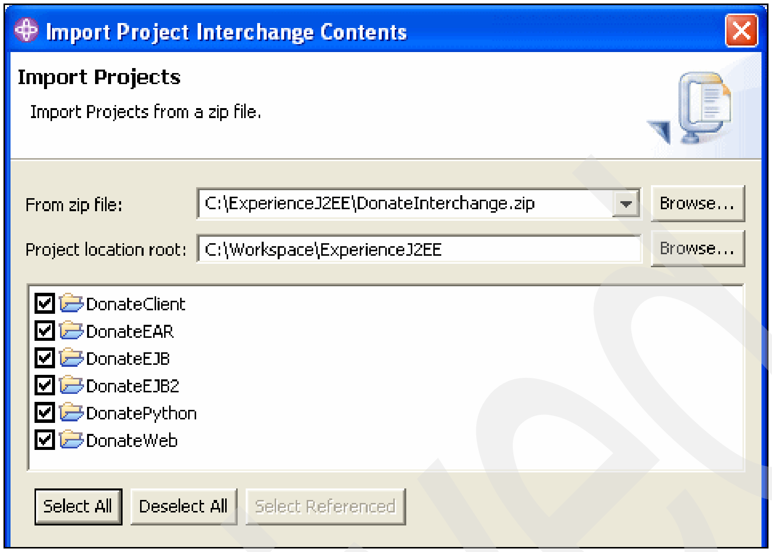
The completed application is provided in the project interchange format, which contains the entire project structure, including metadata files (such as project classpaths and customized JVM settings). This allows for simplified exchange with other AST V6.1 workspaces.

Contrast this with the standard enterprise application archive (EAR) format, which only includes information needed in a J2EE runtime environment and would thus require you to recreate the build environment specific settings.

1. From the AST action bar, select **File** → **Import**.
2. At the **Select** pop-up, select **Project Interchange**, and click **Next**.
3. At the **Import Projects pop-up**, access the following zip files:
   - **From zip file**:

     Windows **C:\ExperienceJ2EE\DonateInterchange.zip**

     Linux **/ExperienceJ2EE/DonateInterchange.zip**
4. Click **Select All**.
5. Click **Finish**.



Import Projects

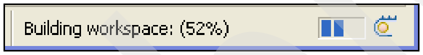
Import Projects from a zip file.

From zip file: C:\ExperienceJ2EE\DonateInterchange.zip

Project location root: C:\Workspace\ExperienceJ2EE

☑ DonateClient
☑ DonateEAR
☑ DonateEJB
☑ DonateEJB2
☑ DonatePython
☑ DonateWeb

6. The activity bar in the lower right tracks the status of the workspace build.



Building workspace: (52%)

7. After the build completes, warnings appear in the **Problems** view (in the lower right pane). These are resolved in the next section.

12.3 Configure the sample application

1. Perform the needed configuration steps for “Chapter 5. Create the employee data access element”:
   - Complete section “5.3.5 Alter the workspace settings to ignore unused methods” on page 101.
   - Complete section “5.4 Configure the test server for DonateEJB” on page 102.
2. Perform the needed configuration steps for “Chapter 6. Create the funds data access element”:
   - Complete section “6.3.5 Alter the workspace settings to ignore specific warnings” on page 148.
   - Complete section “6.4 Create the Funds database and tables” on page 149.

- Complete section “6.5 Configure the test server for DonateEJB2” on page 153.
- Complete section “5.5 Add the DonateEAR application to the test server” on page 114.

> **Behind the scenes**:
>
> This section is intentionally out of order, from a chapter and page number standpoint, from the other sections you are completing.
>
> If you completed this section in order (back in completing the configuration needed for Chapter 5, “Create the employee data access element” on page 83) the publishing of DonateEAR would fail because the resources needed to support DonateEJB2 were not yet defined.

- Complete section “6.7 Test the Funds entity EJB with the UTC” on page 158.

3. Perform the configuration steps needed for “Chapter 7. Create the donate business logic element”:
   - Complete section “7.2.1 Alter the workspace settings to ignore warnings” on page 170.
4. Perform the configuration steps needed for “Chapter 10. Create the application client”:
   - Complete section “10.5 Test the application client” on page 251.
5. Perform the configuration steps needed for “Chapter 11. Implement core security”:
   - Complete section “11.2.1 Create and configure the Jython project and script” on page 261.
   - Complete section “11.2.2 Run the Jython script” on page 262.
   - Complete section “11.3 Enable J2EE application security” on page 265.
   - Complete section “11.6 Update the J2EE application client for security” on page 293.
   - Complete section “11.7 Disable J2EE application security” on page 296.

Part 3

Web services

In Part 3 you extend the core J2EE application to support Web services:

- Chapter 13, “Create the Web service” on page 305: Utilize the AST V6.1 wizards to expose the Donate session EJB bean as a Web service (service), and create a Web service client to access this Web service.
- Chapter 14, “Implement security for the Web service” on page 331: Update the Web service and the Web service client to support Web services security using user ID/password for authentication.

13

Create the Web service

In this chapter you create a Web service that exposes the Donate session EJB.

13.1 Learn!

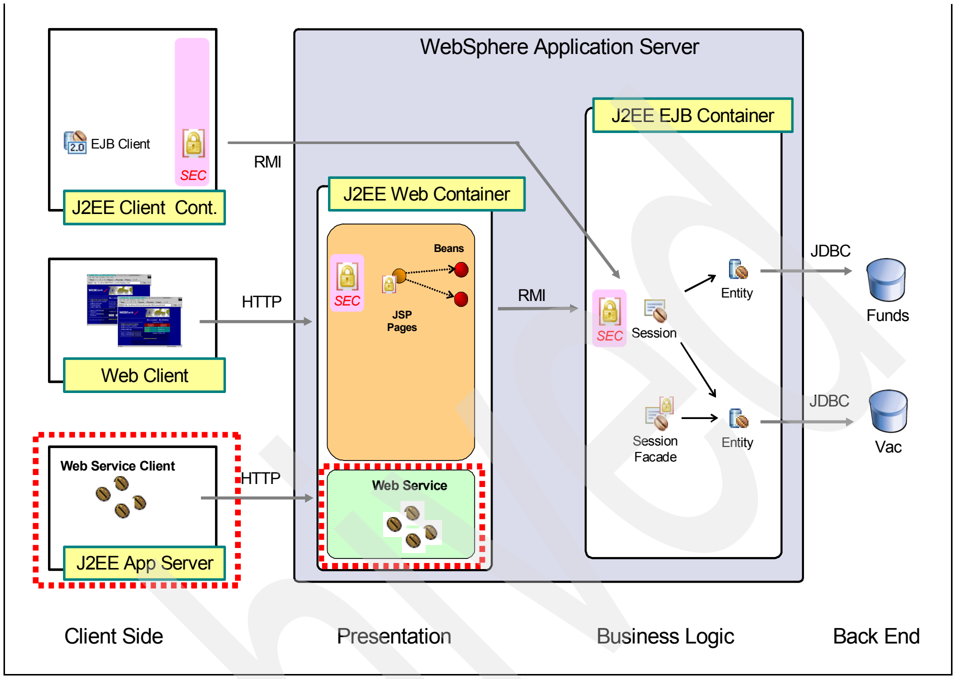


Figure 13-1 Donate application: Web service

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 the following:

- Hyper Text Transfer Protocol (HTTP) provides a common transport protocol to connect the service and the client.

  Some vendors are implementing Simple Object Access Protocol (SOAP) 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 messages describes the remote operation and the associated request or response message data in an XML based format. Runtime environments such as J2EE 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 at most 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), is complex to code to, and potentially can generate a large number of runtime errors.

  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.

13.1.1 Web services and J2EE 1.4

J2EE 1.4 added more specifications to standardize Web services implementation, which improved portability of Web services-based applications between J2EE providers and interoperability with non-J2EE providers. It also provided common programming interfaces, common packaging, and improved interoperability:

- **JSR 101: Java APIs for XML based RPC** (JAX-RPC): Defines a common set of APIs for supporting Web services, providing for a consistent Java-programming interface, for Web services and Web service clients, across J2EE providers.
- **JSR 109: Implementing Enterprise Web services** and **JSR 921: Implementing Enterprise Web services 1.1**: Define a common set of XML-based deployment descriptors, providing for consistent packaging of Web services and Web service clients across J2EE providers.
- **Web services Interoperability** (WS-I): Defines a set of guidelines, which, if followed, provide for enhanced interoperability between Web services implementations, both J2EE and non-J2EE. These guidelines do not replace existing specifications (like SOAP) but refine and clarify their usage to enhance interoperability.

WAS V6.1 ships a Web services engine called **IBM WebSphere** that supports these specifications (or higher). For example, WebSphere Application Server V6.1 supports JAX-RPC 1.1 (exceeding the J2EE V1.4 requirement of JAX-RPC 1.0).

Since these Web services specifications were added in the last several years, there is the requirement to support older Web service implementations that pre-date these specifications. Therefore, WAS V6.1also ships the Apache Axis engine to support those applications that may have different or older requirements.

Table 13-1 Supported Web service engines

Web service engine	WebSphere Application Server versions	Supports		
		JSR-101	JSR-109 / JSR-921	WS-I
IBM WebSphere (recommended)	6.1.x 6.0.x 5.x.	Y	Y	Y
Apache Axis 1.0 (intended for development/testing)	6.1.x 6.0.x 5.x	Y	N	N

Application Server V6.0.x and Application Server V5.x also ship a pre-J2EE 1.4 Web service engine called IBM SOAP.

Additional learn resources are available at the following Web address:

- Java Technology and Web services:

  `http://java.sun.com/webservices/`

- Web Services Interoperability (WSI):

  `http://www.ws-i.org`

- *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461

  `http://www.redbooks.ibm.com/redbooks/pdfs/sg246461.pdf`

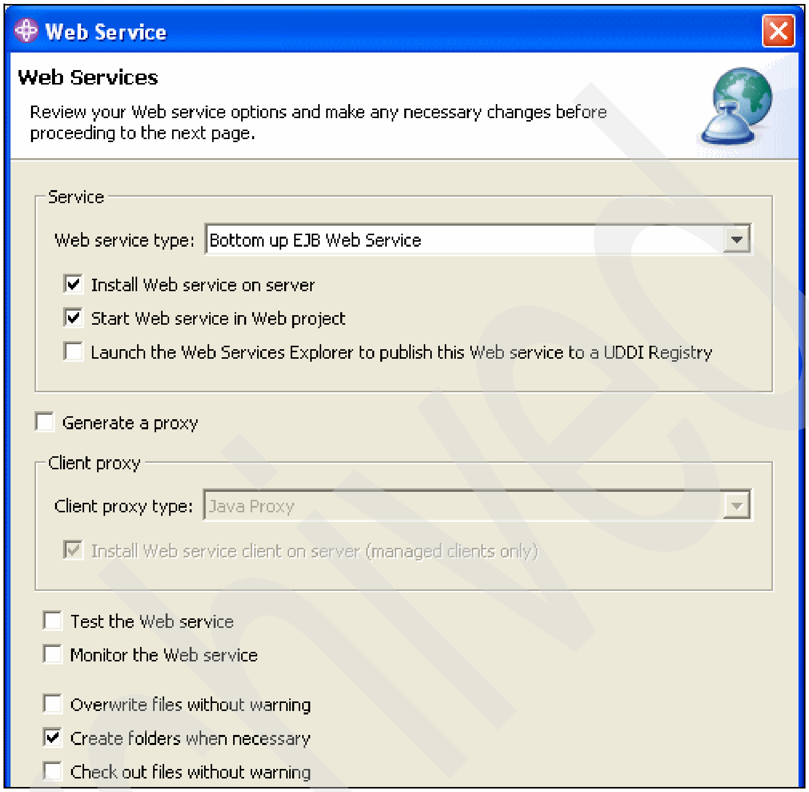
13.2 Create the EJB based Web service

In this section you use the AST V6.1 tooling to create a Web service based on the Donate session EJB.

AST V6.1 also supports creating a Web service based on a standard (non-EJB) JavaBean. However, best practices recommend basing Web services on stateless session EJBs because they can be accessed securely in a consistent manner across all possible interfaces (Web, thick client, EJB, messaging).

1. Ensure that the **ExperienceJ2EE Server @ localhost** test server is running.
2. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Sessions Beans** → **Donate**, and right-click **Web Services** → **Create Web service**.

3. From the **Web Services** popup, ensure that **Web service type** is set to **Bottom up EJB Web Service**, and click **Next**. All other settings default to the selections and values in the following illustration.



Web Service

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Service

Web service type: Bottom up EJB Web Service

☑ Install Web service on server

☑ Start Web service in Web project

☐ Launch the Web Services Explorer to publish this Web service to a UDDI Registry

☐ Generate a proxy

Client proxy

Client proxy type: Java Proxy

☑ Install Web service client on server (managed clients only)

☐ Test the Web service

☐ Monitor the Web service

☐ Overwrite files without warning

☑ Create folders when necessary

☐ Check out files without warning

Behind the scenes:

The **Web service type** selection specifies the base artifact the Web wizard will access.

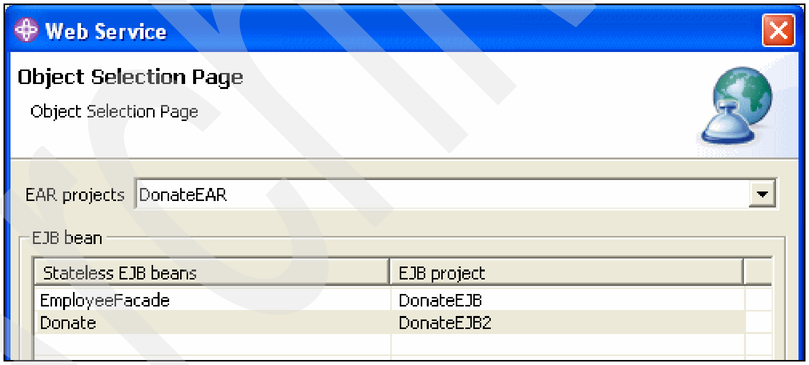
The following selections are supported on WAS V6.1 for creating Web services based on existing artifacts:

- Bottom up Java bean Web service
- Bottom up EJB Web service

The following selections are supported on WAS V6.1 for creating Web service skeletons based on existing Web service definitions (WSDL files). You might use one of these options to implement a *test* version of an existing Web service (developed by someone else) that is not accessible from your current environment.

- Top down Java bean Web service
- Top down EJB Web service

4. From the **Object Selection Page** pop-up, ensure that the **Donate** is selected under **Stateless EJB beans**, and **click Next**.



Object Selection Page

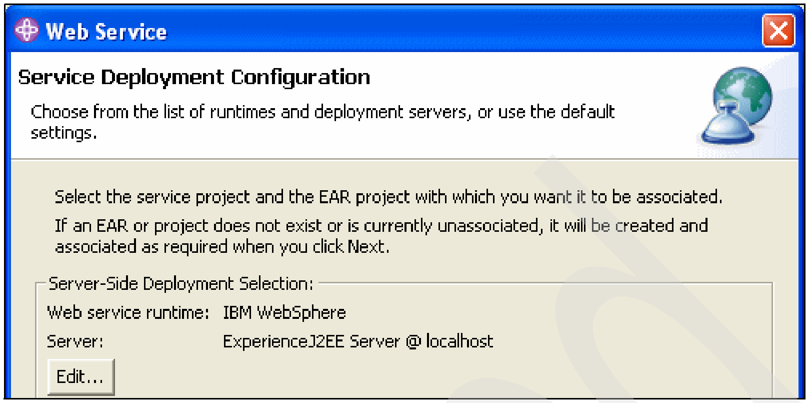
Object Selection Page

EAR projects: DonateEAR

EJB bean

Stateless EJB beans	EJB project
EmployeeFacade	DonateEJB
Donate	DonateEJB2

5. At the **Service Deployment Configuration** pop-up, if **Web service runtime**: is set to **IBM WebSphere** and **Server** is set to **ExperienceJ2EE Server @ localhost**, then skip to step 6; otherwise, click **Edit**.



Web Service

Service Deployment Configuration

Choose from the list of runtimes and deployment servers, or use the default settings.

Select the service project and the EAR project with which you want it to be associated.

If an EAR or project does not exist or is currently unassociated, it will be created and associated as required when you click Next.

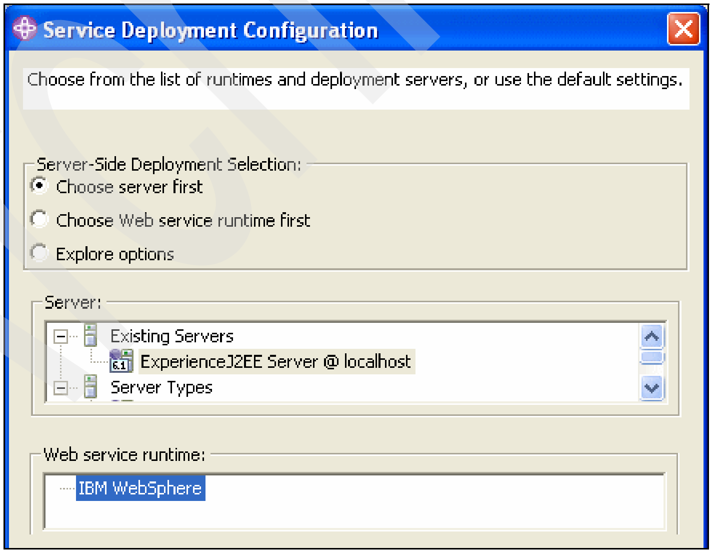
Server-Side Deployment Selection:

Web service runtime: IBM WebSphere

Server: ExperienceJ2EE Server @ localhost

– At the **Choose from the list of runtimes…** pop-up, perform the following tasks:

- Under **Server-Side Deployment Selection**, select **Choose server first**.
- Under **Server**, select **Existing Servers** → **ExperienceJ2EE Server @ localhost**.
- Under **Web Service runtime**, select **IBM WebSphere**.
- Click **OK**.



Service Deployment Configuration

Choose from the list of runtimes and deployment servers, or use the default settings.

Server-Side Deployment Selection:

● Choose server first

○ Choose Web service runtime first

○ Explore options

Server:

Existing Servers

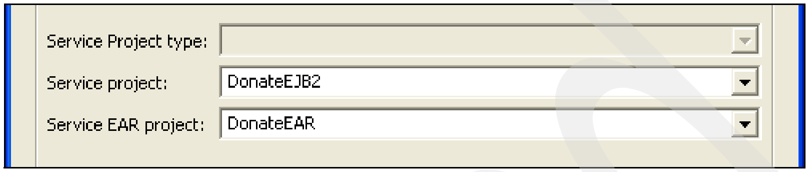
ExperienceJ2EE Server @ localhost

Server Types

Web service runtime:

IBM WebSphere

6. At the **Service Deployment Configuration** pop-up, where **Server** is set to **ExperienceJ2EE Server @ localhost** and **Web service runtime** is set to **IBM WebSphere**, perform the following tasks:
   - **Service project**: **DonateEJB2**
   - **Service EAR project**: **DonateEAR**
   - Click **Next**.



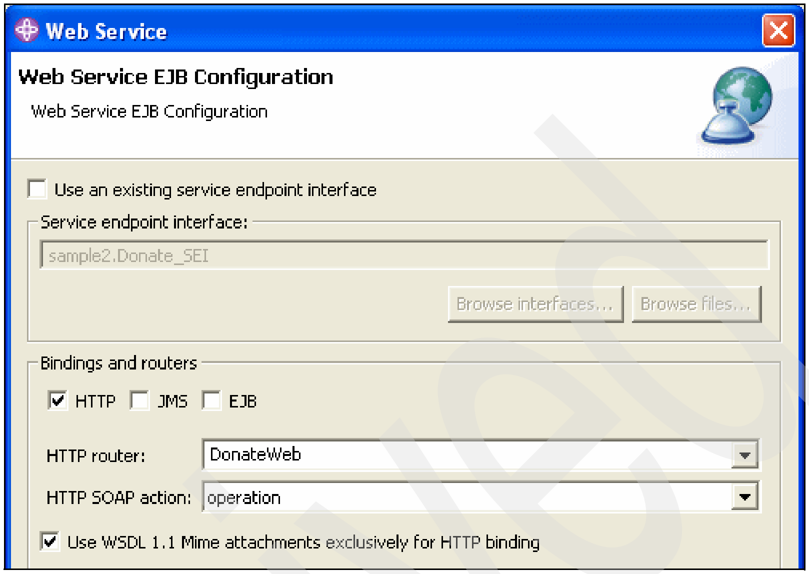
Service Project type:

Service project: DonateEJB2

Service EAR project: DonateEAR

7. At the **Web Services EJB Configuration** pop-up, perform the following tasks:
   - Ensure that **Use an existing service endpoint interface** is cleared.
   - Under **Bindings and routers**, perform the following tasks:
     - Ensure that **HTTP** is selected (and **JMS** and **EJB** are cleared).
     - **HTTP Router**: **DonateWeb**
     - **HTTP SOAP action**: **operation**
     - Click **Next**.

All other settings should default to the selections and values shown in the following illustration.


Web Service

Web Service EJB Configuration

Web Service EJB Configuration

□ Use an existing service endpoint interface

Service endpoint interface: sample2.Donate_SEI

Bindings and routers

☑ HTTP □ JMS □ EJB

HTTP router: DonateWeb

HTTP SOAP action: operation

☑ Use WSDL 1.1 Mime attachments exclusively for HTTP binding

Behind the scenes:

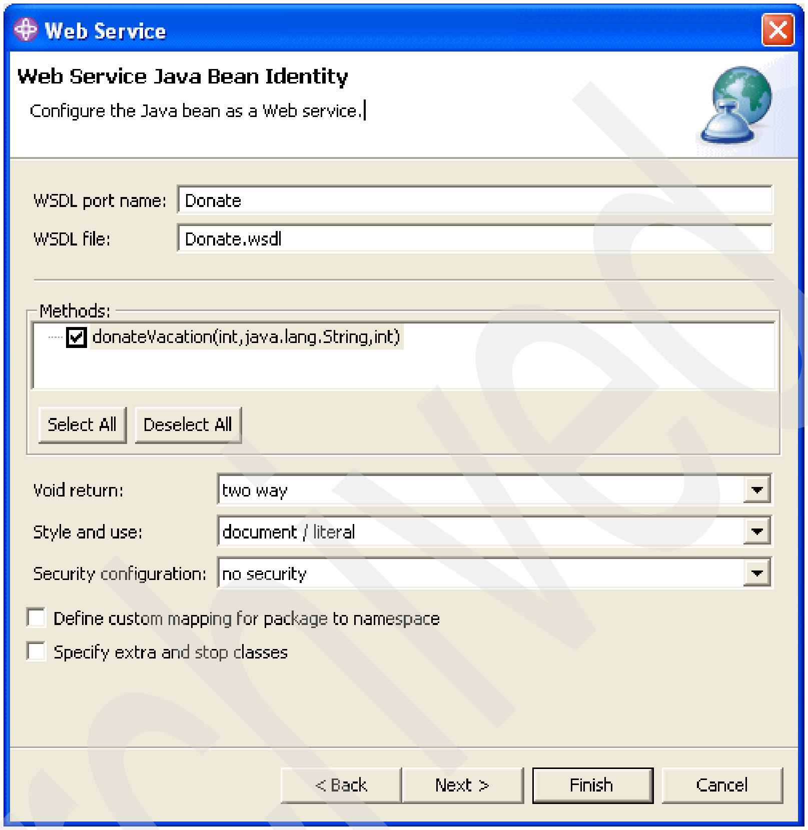
- **Bindings and routers**: The JAX-RPC specifications provide for a binding only over the HTTP transport, which is handled or routed through a servlet in a Web project. IBM implemented the following extensions:
  - JMS, which uses a message-driven bean (in an EJB project) to receive a JMS message and then invoke a session EJB.
  - EJB, which uses the RMI communication to directly access a session EJB (in an EJB project).

  Note that a single Web service can be made available over one or more of these bindings.

- **HTTP SOAP action**: Sets the text used in the soapAction field in the WSDL file. Note that a WSDL can contain multiple actions, with each action mapping to a single method in the supporting Java artifact. The recommendation on this pop-up is to use the **operation** value (the default for this field), which uses the name of the method. In this case, the following wsdl operation stanza is generated:

```
<wsdl:operation name="donateVacation">
       <wsdlsoap:operation soapAction="donateVacation"/>
```

8. At the **Web Service Java Bean Identity** pop-up, ensure that **donateVacation** is selected, and click **Next**. All other settings should default to the selections and values shown in the following illustration.



Web Service

Web Service Java Bean Identity

Configure the Java bean as a Web service.

WSDL port name: Donate

WSDL file: Donate.wsdl

Methods:

☑ donateVacation(int,java.lang.String,int)

Void return: two way

Style and use: document / literal

Security configuration: no security

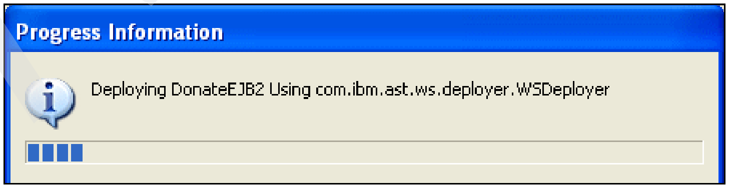
☐ Define custom mapping for package to namespace

☐ Specify extra and stop classes

Behind the scenes:

WS-I compliance requires using **document/literal** (under **Style and use**).

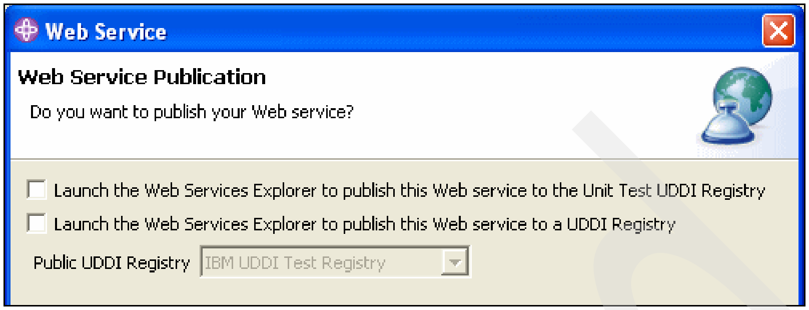
9. Wait for the **Operation In Progress…** pop-up to complete.



Progress Information

Deploying DonateEJB2 Using com.ibm.ast.ws.deployer.WSDeployer

10. At the resulting **Web Service Publication** pop-up, click **Finish**. Do not publish to any of the UDDI registries.



Web Service

Web Service Publication

Do you want to publish your Web service?

□ Launch the Web Services Explorer to publish this Web service to the Unit Test UDDI Registry

□ Launch the Web Services Explorer to publish this Web service to a UDDI Registry

Public UDDI Registry: IBM UDDI Test Registry

Behind the scenes:

AST V6.1 allows you to optionally publish the WSDL to a UDDI registry so that others can use it to access and test your Web service:

- **Unit Test UDDI Registry**: A UDDI V3 compliant implementation provided with AST V6.1. It has a Web interface and can also be accessed using the AST V6.1 Web Services Explorer, the UDDI Web service interface, or the UDDI EJB interface. This provides sufficient capability for a developer to test applications that utilize the UDDI V3 APIs.
- **Public UDDI Registry**: These publicly available UDDI sites are in the process of being terminated.

  The following excerpt comes from the following Web site:

  http://www.ibm.com/software/solutions/webservices/uddi

Version 3 of the Universal Description, Discovery and Integration (UDDI) V3 was approved as an OASIS standard in February 2005. Having achieved that milestone, the companies hosting the UDDI Business Registry (UBR) evaluated the results of hosting a reference implementation for the UDDI technology. Since the fall of 2000, the UBR has been a significant asset for testing during the development of the three versions of the UDDI specifications and the vendor software based on it. UBR also provided a proof of concept for early Web services business applications. Considering that Web service and UDDI specifications are now mature standards and the significant number of vendor supplied UDDI products that are available, hosting this registry is no longer necessary..... Over the past five years, as Web services applications have matured, the role of UDDI based registries has also evolved in part based on testing experiences with UBR. Registries based on UDDI have been established within enterprises and organizations and have an important role in Web services business applications. The availability of UBR provided valuable validation and guidance during the early days of UDDI. With the fulfillment of these goals, the UBR was discontinued.

The value and success (or not) of UDDI is the source of heated debate in the overall Web services community, and any attempt to summarize all sides of this debate would extend for many pages.

The important summary for the Experience! phase is that UDDI is an optional Web services component.

Behind the scenes:

The following describes some, but not all, of the files and configuration changes created by the wizard:

- **Dynamic Web Projects → DonateWeb → WebContent → WEB-INF → web.xml** was updated to include a servlet that handles the Web service requests. The Web services Servlet class utilizes the subsequent configuration files to process the inbound Web services requests.

```
<servlet>
   <display-name>
   Web Services Router Servlet</display-name>
   <servlet-name>Donate</servlet-name>
   <servlet-class>

com.ibm.ws.webservices.engine.transport.http.WebServicesServlet
   </servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>Donate</servlet-name>
   <url-pattern>services/Donate</url-pattern>
</servlet-mapping>
```

- **EJB Projects → DonateEJB2 → ejbModule → META-INF → wsdl → Donate.wsdl** contains the definition that others can use to create a Web service client that accesses this Web service.
  - Note the wsdlsoap address near the bottom of the WSDL. This URL points to the Web service servlet (services/Donate) shown below.

```
<wsdl:port binding="intf:DonateSoapBinding" name="Donate">
   <wsdlsoap:address location=
      "https://localhost:9443/DonateWeb/services/Donate"/>
</wsdl:port>
```

  - Note that the WSDL file is created in the EJB project, which contains the business logic behind the Web service, and not in the Web project that contains the servlet that exposes the Web service.

The EJB project location is sufficient for your development activities since the Web Services Explorer and various wizards can locally access the file; however, it may not be sufficient for a runtime environment because the WSDL is not accessible over HTTP, and others may wish to look at the WSDL without using a UDDI registry.

In that case, you could manually copy the WSDL from the EJB project to the Web project (typically into <web project> → WebContent → wsdl).

- **EJB Modules** → **DonateEJB2** → **ejbModule** → **META-INF** → **webservices.xml** links the Donate Web service to the Donate session EJB:

```
<webservice-description>
   <webservice-description-name>DonateService
   </webservice-description-name>
   <wsdl-file>META-INF/wsdl/Donate.wsdl</wsdl-file>
   <jaxrpc-mapping-file>META-INF/Donate_mapping.xml
   </jaxrpc-mapping-file>
   <port-component>
      <port-component-name>Donate</port-component-name>
      <wsdl-port xmlns:pfx="http://sample2">pfx:Donate
      </wsdl-port>
      <service-endpoint-interface>sample2.Donate_SEI
      </service-endpoint-interface>
      <service-impl-bean>
         <ejb-link>Donate</ejb-link>
      </service-impl-bean>
   </port-component>
</webservice-description>
```

- **EJB Modules** → **DonateEJB2** → **ejbModule** → **META-INF** → **Donate_mapping.xml** describes how to map the parameters to and from the Web services XML format and the Java format utilized by the EJB.
- **EJB Modules** → **DonateEJB2** → **ejbModule** → **META-INF** → **ibm-webservices-bnd.xmi** contain any IBM-unique extensions needed for Web service security. In this case no significant information since your Web service is running without Web service security.

13.3 Modify the WSDL to remove HTTPS

The WSDL is generated using the secure https://localhost:9443 because AST V6.1 is configured to interact with the test server in secure mode.

The Web service can be accessed over the HTTPS or HTTP. Since the HTTPS and HTTP interaction is handled by the embedded HTTP server in the Web container, the back end Web service code is unaware of which protocol is used.

However, you can run into two issues in this book if you left the WSDL configured with HTTPS:

- The Web Service Explorer generates the following exception if you attempt to access a Web service over HTTPS:

  ```
  IWAB0135E An unexpected error has occurred.
  IOException
  Cannot find the specified class
  java.security.PrivilegedActionException:
  java.lang.ClassNotFoundException:
  com.ibm.websphere.ssl.protocol.SSLSocketFactory
  ```

  This is a known problem in AST V6.1 and is identified as defect 368871, which should be included in a future AST V6.1 fixpack.

- The wizard that generates the Web service client, which you use in section “13.5.2, Create and test the Web service client.” on page 324, does not generate the required SSL configuration statements. Therefore, access using an unmodified client fails with the following error:

  ```
     Result
     exception:
     com.ibm.wsspi.channel.framework.exception.ChannelException:
     com.ibm.wsspi.channel.framework.exception.ChannelException:
     Invalid trust file name of null
  ```

  The SSL setup can be time consuming and is not the major focus of this scenario.

Therefore, follow these steps to change the WSDL address from HTTPS to HTTP:

1. Select **EJB Projects** → **DonateEJB2** → **ejbModule** → **META-INF** → **wsdl** → **Donate.wsdl**, and double-click to open.
2. At the **Donate.wsdl** editor switch to the **Source** tab, scroll to the bottom, and change the SOAP address line from the following

   ```
   <wsdlsoap:address
   location="https://localhost:9443/DonateWeb/services/Donate"/>
   ```

to the following, ensuring that you change BOTH https to http AND 9443 to 9080:

```
<wsdlsoap:address
location="http://localhost:9080/DonateWeb/services/Donate"/>
```

3. Save and close.

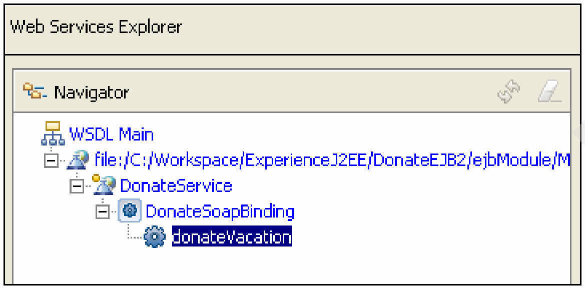
13.4 Test the Web service using the Web Services Explorer

AST V6.1 contains a generic test tool called the Web Services Explorer that enables you to load a WSDL file, either within AST V6.1 or available anywhere over HTTP, and manually test the referenced Web service.

1. From the Project Explorer, select **EJB Projects** → **DonateEJB2** → **ejbModule** → **META-INF** → **wsdl** → **Donate.wsdl**, and right-click **Web Services** → **Test with Web Services Explorer**.

> **Alternative**:
>
> The previous step launches the Web Services Explorer and opens the test page for Donate.WSDL. You can also access this page using the following instructions:
>
> 1. From the AST V6.1 action bar, right-click **Run** → **Launch Web Services Explorer**.
> 2. At the **Web Services Explorer** browser, click the **WSDL Page** icon [ ] in the upper right.
> 3. In the **Navigator** pane, select **WSDL Main**, and then on the resulting **Actions** pane click **Browse…**
> 4. At the resulting **WSDL Browser** pop-up perform the following actions:
>   - **Select a WSDL source**: **Workspace Projects**
>   - **Workspace Project**: **DonateEJB2**
>   - **WSDL URL**: Select **platform:/resource/DonateEJB2/ejbModule /META-INF/wsdl/Donate.wsdl** from the pull-down.
>   - Click **Go**.
> 5. At the **Actions** pane, with the **WSDL URL** populated with the value entered in step 4, click **Go**.

6. In the **Navigator** pane, select **WSDL Main** → **file:/C:/Workspace...** → **DonateService** → **DonateSoapBinding** → **donateVacation**.

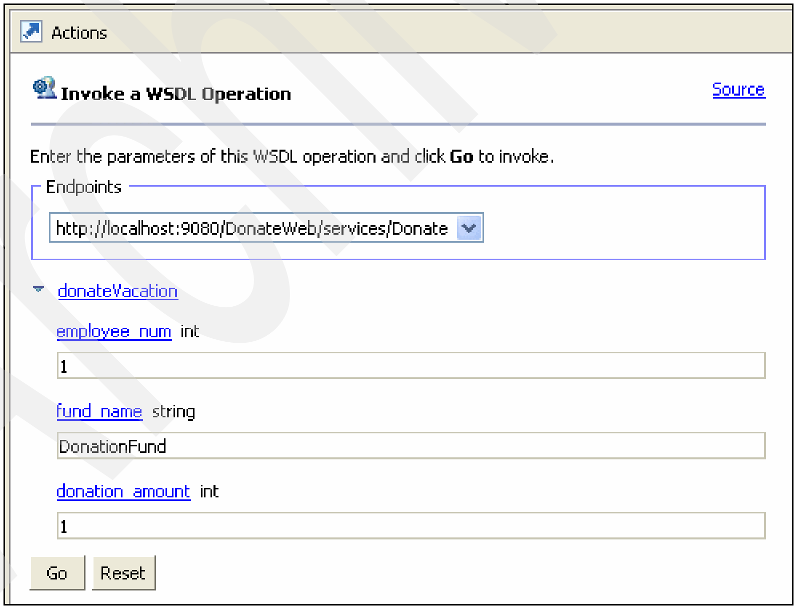


Web Services Explorer

Navigator

- WSDL Main
  - file:/C:/Workspace/ExperienceJ2EE/DonateEJB2/ejbModule/M
    - DonateService
      - DonateSoapBinding
        - donateVacation

7. In the **Actions** pane, which is now pointing to **Invoke a WSDL Operation**, complete the following fields:
   - **employee_num**: **1**
   - **fund_name**: **DonationFund**
   - **donation_amount**: **1**
8. Click **Go**.



Actions

Invoke a WSDL Operation

Source

Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints

http://localhost:9080/DonateWeb/services/Donate

donateVacation

employee_num int

1

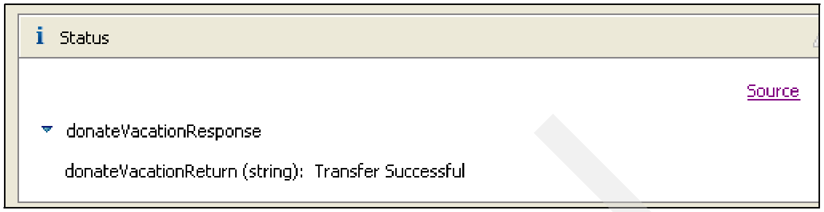
fund_name string

DonationFund

donation_amount int

1

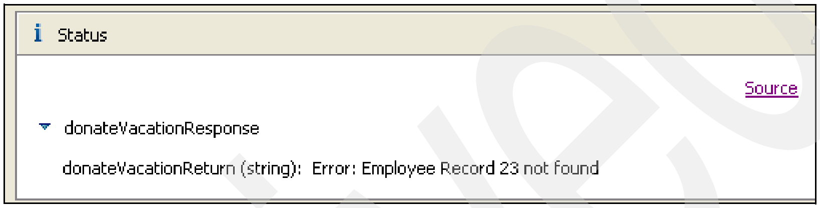
Note the message in the **Status** pane (lower right), indicating a successful result:



donateVacationResponse

donateVacationReturn (string): Transfer Successful

9. On the **Actions** pane try invoking the Web service with an employee_num of 23. Note the error message in the **Status** pane.



donateVacationResponse

donateVacationReturn (string): Error: Employee Record 23 not found

10. Close the **Web Services Explorer**.

13.5 Create and test a Web service client

Other systems use the WSDL file, which the Web service wizard creates, (J2EE or non-J2EE) to generate a client that can access the Web service. In this section you create and test a Web service client that can run on WAS V6.1.

13.5.1 Create a new enterprise application project

In this section you create a second enterprise application project that contains the generated Web service client:

1. From the **Project Explorer**, select **Enterprise Applications**, and right-click **New** → **Enterprise Application Project**.
1. At the **Enterprise Application Project** pop-up, set the **Project Name** to **DonateWSClientEAR**, and click **Next**.
2. At the **Select Project Facets** pop-up, click **Next**.
3. At the **J2EE Modules to Add to the EAR** pop-up, click **New Module…**.
4. At the resulting **New Module Project** pop-up, perform the following actions:
   - Clear **Application client module, EJB module,** and **Connector module**.

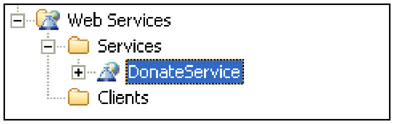
- Set the **Web module** to **DonateWSClientWeb**.
- Click **Finish**.

5. Back at the **J2EE Modules to Add to the EAR** pop-up, ensure that **DonateWSClientWeb** is selected, and click **Finish**.

13.5.2 Create and test the Web service client.

In this section you use the AST V6.1 tooling to create a Web service client from the existing Donate.wsdl.

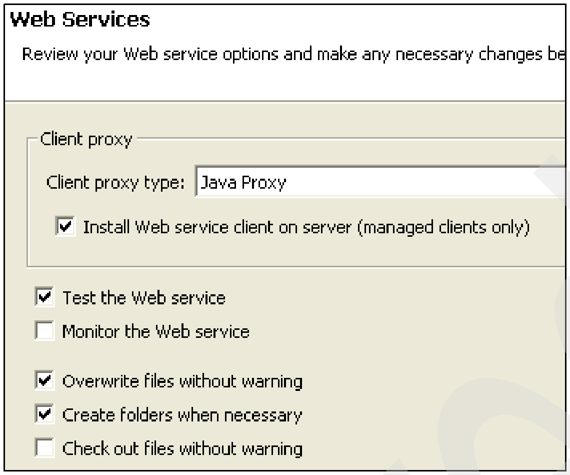
1. From the **Project Explorer**, select **Web Services** → **Services** → **DonateService**, and right-click **Generate Client**.



Behind the scenes:

AST V6.1 allows you to create, view, and edit the Web service definitions, both Services and Clients, through the Web Services grouping in the J2EE perspective Project Explorer. This provides an abstract view of all the Web services contained in the workspace.

2. At the **Web Services** pop-up, perform the following actions:
   - Ensure that the **Client proxy type** is set to **Java Proxy**.
   - Ensure that **Install Web service client on server (managed clients only)** is selected.
   - Select **Test the Web service**.
   - Click **Next**.



Web Services

Review your Web service options and make any necessary changes be

Client proxy

Client proxy type: Java Proxy

☑ Install Web service client on server (managed clients only)

☑ Test the Web service

□ Monitor the Web service

☑ Overwrite files without warning

☑ Create folders when necessary

□ Check out files without warning

Behind the scenes:

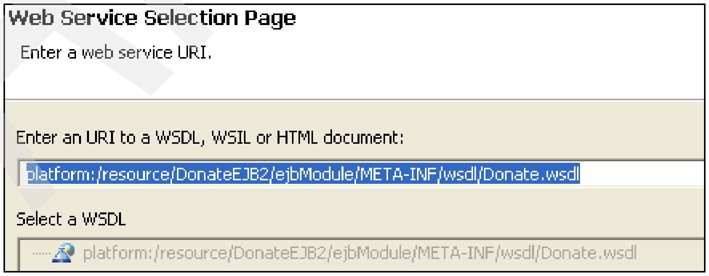
The **Install Web service client on server (managed clients only)** option instructs the wizard to automatically add and Publish the Enterprise Application project to the test server.

The **Test the Web service** option indicates that you want to test the Web service client either through the Web Services Explorer or by generating a test client.

3. At the **Web Service Selection Page** pop-up, note that the URI to the WSDL is set to the following:

   platform:/resource/DonateEJB2/ejbModule/META-INF/wsdl/Donate.wsdl

4. Click **Next**.



Web Service Selection Page

Enter a web service URI.

Enter an URI to a WSDL, WSIL or HTML document:

platform:/resource/DonateEJB2/ejbModule/META-INF/wsdl/Donate.wsdl

Select a WSDL

platform:/resource/DonateEJB2/ejbModule/META-INF/wsdl/Donate.wsdl

5. At the **Client Environment Configuration** pop-up, if **Web service runtime**: Is set to **IBM WebSphere** and **Server**: Is set to **ExperienceJ2EE Server @ localhost** then skip to step 7; Otherwise, click **Edit** and continue to step 6:

6. At the **Choose from the list of runtimes…** pop-up, perform the following actions:

   – Under **Client-Side Environment Selection**, select **Choose server first.**

   – Under **Server**, select **Existing Servers** → **ExperienceJ2EE Server @ localhost.**

   – Under **Web Service runtime**, select **IBM WebSphere**.

   – Click **OK**.

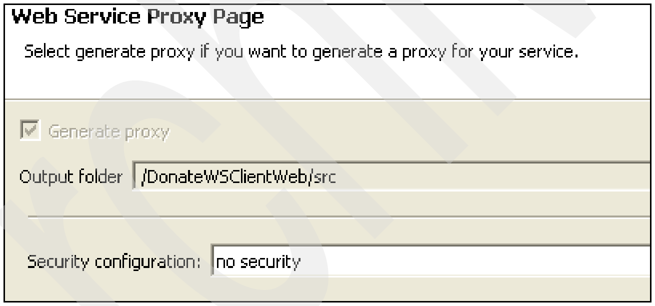
7. Back at the **Client Environment Configuration** pop-up, where **Server** is set to **ExperienceJ2EE Server @ localhost** and **Web service runtime** is set to **IBM WebSphere**, perform the following actions:

   – **Client project**: **DonateWSClientWeb**

   – **Client EAR project**: **DonateWSClientEAR**

   – Click **Next**.

8. At the **Web Service Proxy Page** pop-up, ensure that **Security Configuration** is set to **no security**, and click **Finish**.



Web Service Proxy Page

Select generate proxy if you want to generate a proxy for your service.

☑ Generate proxy

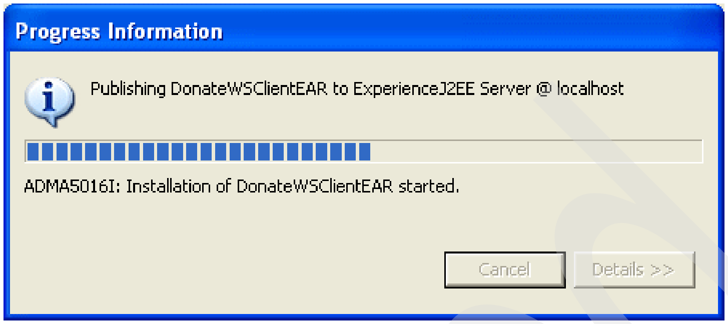
Output folder: /DonateWSClientWeb/src

Security configuration: no security

Off course?

If you do not see the Security Configuration option mentioned in step 8, your Web service engine is incorrectly set to Apache Axis 1.0. It should be set to IBM WebSphere. This incorrect setting prevents you from updating the Web service to use security (in Chapter 14, “Implement security for the Web service” on page 331) because the Apache Axis 1.0 engine does not support the WS-Security specification.

A **Progress Information** popup appears, indicating that DonateWSClientEAR is automatically being added to the test server and started. This pop-up exits automatically if the publish succeeds.

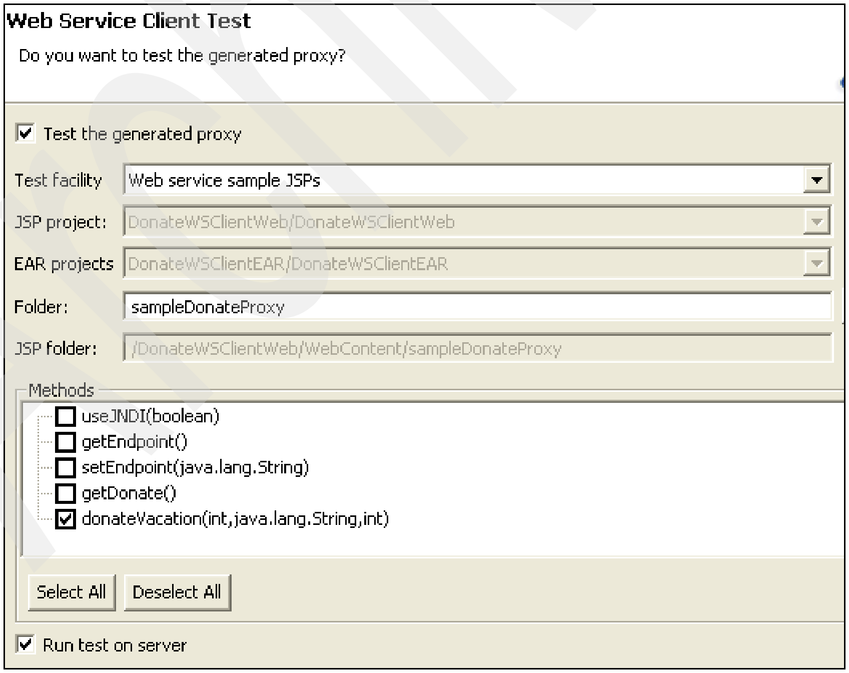


Progress Information

Publishing DonateWSClientEAR to ExperienceJ2EE Server @ localhost

ADMA5016I: Installation of DonateWSClientEAR started.

9. At the **Web Service Client Test** pop-up, perform the following actions:
   - Select **Test the generated proxy**.
   - **Test Facility**: **Web service sample JSPs**.
   - Clear all methods except **donateVacation**.
   - Select **Run test on server**.
   - Click **Finish**.



Web Service Client Test

Do you want to test the generated proxy?

☑ Test the generated proxy

Test facility: Web service sample JSPs

JSP project: DonateWSClientWeb/DonateWSClientWeb

EAR projects: DonateWSClientEAR/DonateWSClientEAR

Folder: sampleDonateProxy

JSP folder: /DonateWSClientWeb/WebContent/sampleDonateProxy

Methods

□ useJNDI(boolean)
□ getEndpoint()
□ setEndpoint(java.lang.String)
□ getDonate()
☑ donateVacation(int,java.lang.String,int)

☑ Run test on server

10. The wizard automatically publishes **DonateWSClientEAR** to the **ExperienceJ2EE Server@localhost test server** and launches the generated test client using the following Web address:

```
https://localhost:9443/DonateWSClientWeb/sampleDonateProxy/TestClien
t.jsp
```

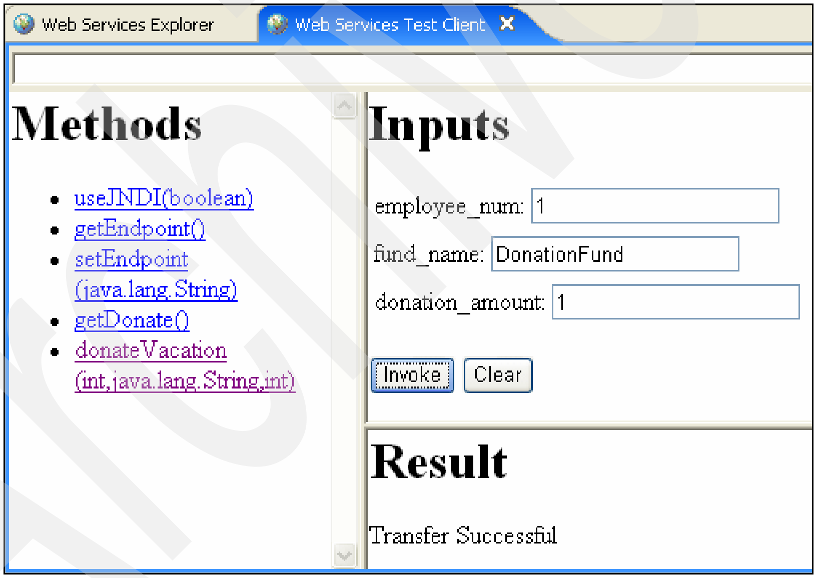
11. At the resulting **Web Services Test Client** browser, perform the following actions:

   – In the **Methods** pane, select **donateVacation**.

   – In the **Inputs** pane:

   - **employee_num**: **1**
   - **fund_name**: **DonationFund**
   - **donation_amount**: **1**
   - Click **Invoke**.

   – In the **Results** pane, view the result.



employee_num: 1

fund_name: DonationFund

donation_amount: 1

Transfer Successful

   – Close the **Web Services Explorer** and the **Web Services Test Client**.

13.6 Explore!

The Web service and Web service client were created using base Web service definitions and did not address the evolving standards both in Web services and in J2EE.

Many WS specifications (driven by the WS-I consortium) are under development and consideration by the broader Web service community. Some, like WS-Security, are well established and included with the IBM WebSphere Application Server (for quite some time). Others are fairly recent and were added with Application Server V6.0 or V6.1:

- WS-Addressing
- WS-Resource Framework
- WS-Coordination
- WS-Atomic
- WS-BusinessActivity
- WS-Notification

However, additional WS specifications are continuing to be finalized, and the Java EE specification was released in May, 2006 with enhanced J2EE specifications that are needed to support some of these newer WS specifications.

Therefore, IBM announced plans to deliver an update to WAS V6.1 called the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services. IBM also has a beta version currently available—see the following resource links.

This feature pack will include the following new WS standards:

- WS-Reliable Messaging (WS-RM)
- WS-Addressing (WS-Addressing)
- SOAP Message Transmission Optimization Mechanism (MTOM)

It also includes the following updated J2EE and other development standards:

- Java API for XML Web Services (JAX-WS 2.0)
- Java Architecture for XML Binding (JAXB 2.0)
- SOAP with Attachments API for Java (SAAJ 1.3)
- Streaming API for XML (StAX 1.0)

In combination, these specifications provide an early implementation of the WS-Interoperability Reliable Secure Profile that is currently under development.

Additional explore resources are available at the following Web address:

- IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services:

`https://www14.software.ibm.com/iwm/web/cc/earlyprograms/websphere/wsvwas61/?S_TACT=105AGX28&S_CMP=DLMAIN`

- *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257, located at the following Web address:

  `http://www.redbooks.ibm.com/redpieces/abstracts/sg247257.html`

14

Implement security for the Web service

In this chapter you update the Web service and Web service client from the previous chapter to utilize WS-Security.

14.1 Learn!

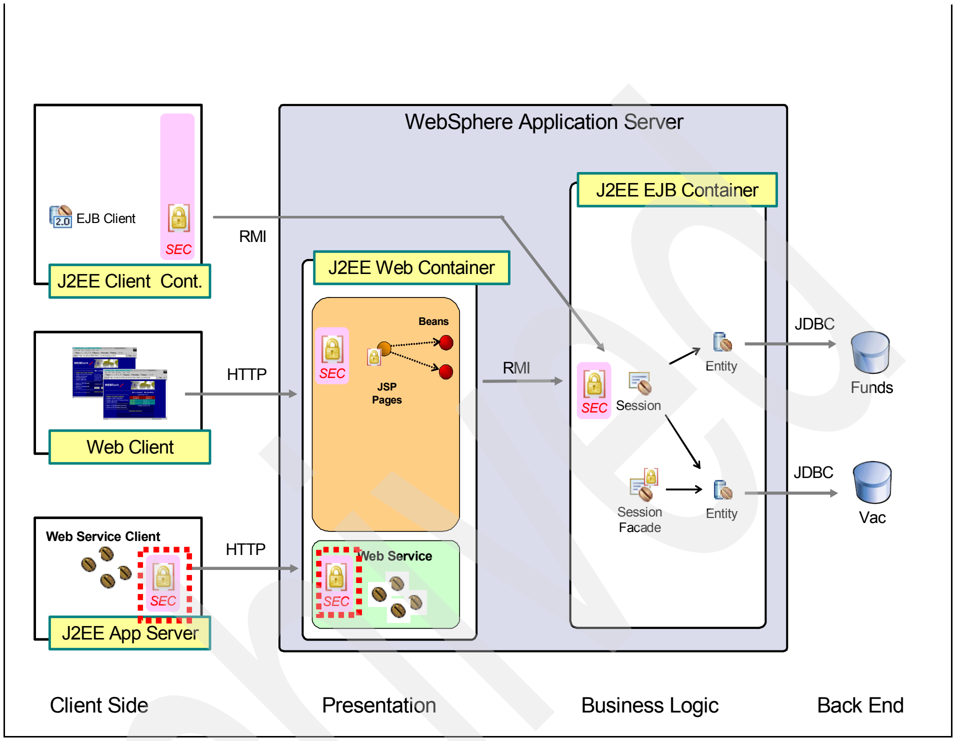
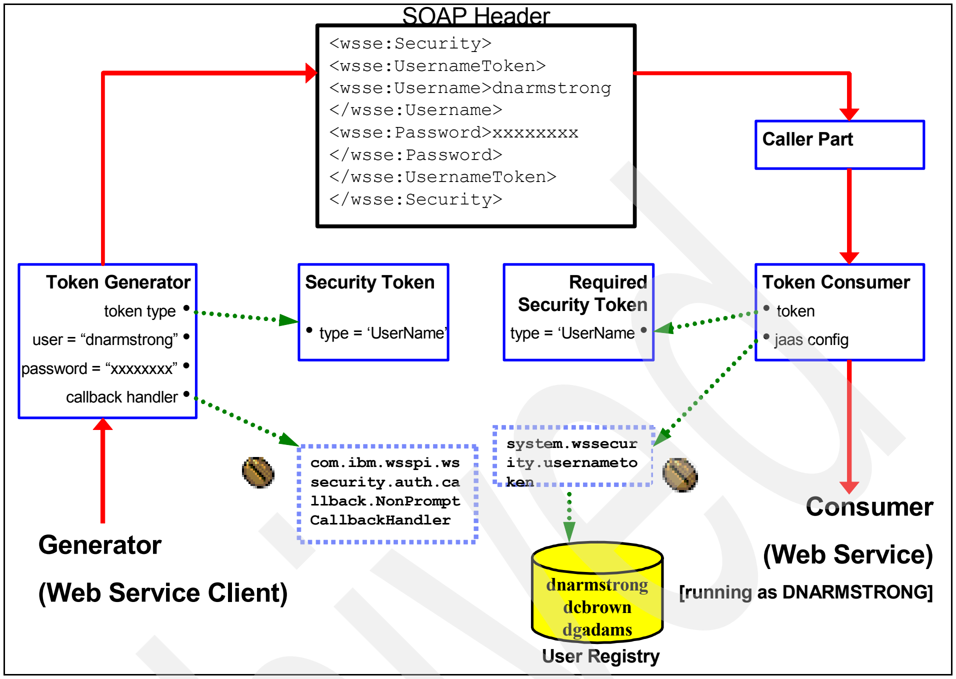


Figure 14-1 Donate application: Web service security

The original Web services specification did not address security. Web services providers and consumers had to manually implement security mechanisms using techniques like encrypting the message payloads and running the Web services over HTTPS. Although these techniques are effective, they still represent user-written code and increase the complexity of utilizing Web services.

Competing specifications were developed for Web services security, but in recent years the WS-Security specification—supported by IBM, Microsoft, and Sun among others—became the most widely adopted, and this specification is now managed by the Web Services Interoperability Organization.

Figure 14-2 on page 333 shows the configuration variations of the elements required in the Web service client and in the Web service deployment descriptors to ensure that a username and password are put into the wsse:Security element in the SOAP header (in the SOAP request) and that it is processed by the Web service engine.



```
<wsse:Security>
<wsse:UsernameToken>
<wsse:Username>dnarmstrong
</wsse:Username>
<wsse:Password>xxxxxxxx
</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
```

Figure 14-2 Web service security implementation

In the Web service client deployment descriptors, the following applies:

- The Security Token defines the type of security token—in this case Username.
- The Token Generator causes the wsse:Security element to be generated.
  - The Security Token field defines this as type Username.
  - The user and password fields provide the input username and password.
  - The callback handler of com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler. indicates that the Token Generator gets the user and password from the configuration files and does not prompt the user.

In the Web service deployment descriptors, the following applies:

- The Required Security Token element indicates that a security token of type Username is required.
- The Caller Part element indicates that this token comes from the inbound message.

- The Token Consumer element describes how the username/password is processed.
  - The Token consumer class field indicates that the token is read by the com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer class.
  - The Security Token field identifies the information as coming from the Required Security Token information, which comes from the Caller Part.
  - The jaas.config.name field indicates that the login to J2EE is processed by the system.wssecurity.UsernameToken class.

As a result of these descriptors, the username and password specified on the Web service client side are used on the Web service side to log into the J2EE application server, and the session EJB runs under that user identity.

The following additional learn resources is available:

- See “Chapter 8. Web services security,” in *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257, located at the following Web address:

  http://www.redbooks.ibm.com/abstracts/sg247257.html

14.2 Re-enable J2EE application security

In this section you perform the complete set of tasks needed to switch a test server and workspace combination from running with J2EE application security disabled to a combination running with J2EE application security enabled.

14.2.1 Re-enable J2EE application security

You only have to enable the J2EE application security, since ExperienceJ2EE Server @ localhost already has a configured repository and already has J2EE administrative security enabled.

1. Start the **Admin Console** using the same steps as described in section “5.4.1 Start the admin console” on page 102.
   - Recall that at the **Admin Console** login screen you use the **j2eeadmin** user ID and password that you defined in section “3.5 Create the ExperienceJ2EE application server profile” on page 51.
2. On the left expand to **Security**, and click **Secure administration, applications, and infrastructure**.
3. On the right, in **Secure administration, applications, and infrastructure**, perform the following actions:

   - Under **Application security**, select **Enable application security**.
   - Click **Apply**.
4. At the top of the panel, in the **Messages** box that states “**Changes have been made to your local configuration….**”, click **Save**.
5. Logout and close the **Admin Console**.

14.2.2 Restart the ExperienceJ2EE Server @ localhost test server

1. In AST V6.1, switch to the **Servers** view in the lower-right pane, select **ExperienceJ2EE Server @ localhost**, and select the re-start icon [ ] in the action bar. 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
   ```

14.3 Test the unsecured Web service

In this section you test the existing *unsecured* Web service and Web service client to verify that they fail with J2EE application security enabled.

1. From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWSClientWeb** → **WebContent** → **sampleDonateProxy** → **TestClient.jsp**, and right-click **Run As** → **Run on Server**.
   - From the resulting **Define a New Server** pop-up, perform the following actions:
     - Select **Choose an existing server**.
     - Under **Select the server that you want to use**, select **localhost** → **ExperienceJ2EE Server @ localhost**.
     - Optionally select **Set server as project default (do not ask again)**.
     - Click **Finish**.
2. At the resulting **Web Services Test Client** browser, perform the following actions:
   - In the **Methods** pane, select **donateVacation**.
   - In the **Inputs** pane, perform the following actions:
     - **employee_num**: **1**
     - **fund_name**: **DonationFund**

- **donation_amount**: **1**
- Click **Invoke**.

– Note the following error message in the **Result** pane:

```
exception: java.rmi.AccessException: ; nested exception is:
com.ibm.websphere.csi.CSIAccessException: SECJ0053E:
Authorization failed for /UNAUTHENTICATED while invoking
(Bean)ejb/sample2/DonateHome
donateVacation(int,java.lang.String,int):5 securityName:
/UNAUTHENTICATED;accessID: UNAUTHENTICATED is not granted any of
the required roles: DUSERS_ROLE DMANAGERS_ROLE
```

3. Close the **Services Test Client** browser.
4. Optionally examine the **Console** view for **ExperienceJ2EE Server @ localhost** to see a similar error message.

Behind the scenes:

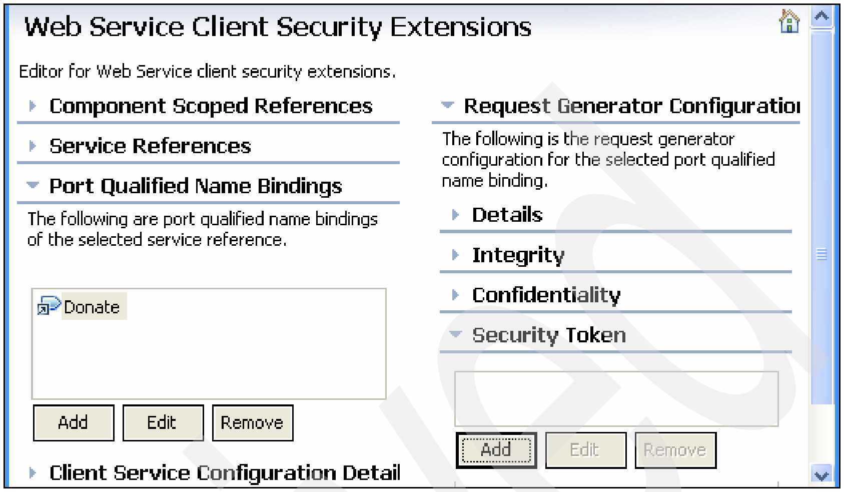
The access fails because the Web service request is received without authentication, and the security constraint in DonateEJB2 ejb-jar.xml, which you updated to apply to all interfaces, blocks the unauthenticated access.

14.4 Update the Web service client to support secure access

In this section you update the Web service client deployment descriptors to include the username and password.

1. Add the security token by performing the following actions:
   – From the **Project Explorer**, select **Web Services** → **Clients** → **DonateWSClientWeb**, and double-click to open the **Web Deployment Descriptor**.
   – At the resulting **Web Deployment Descriptor** editor, switch to the **WS Extension** tab.
   – On the left, expand **Port Qualified Name Bindings**, and select **Donate**.

- On the right, expand to **Request Generator Configuration** → **Security Token**, and click the **Add** button located on the right, under the Security Token section.

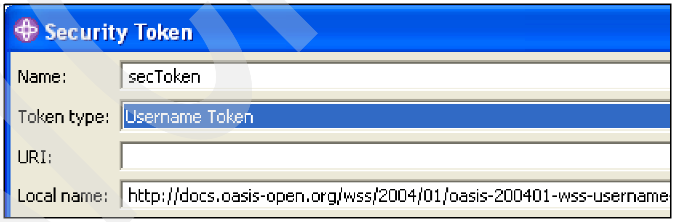

Web Service Client Security Extensions
Component Scoped References
Service References
Port Qualified Name Bindings
Donate
Add
Edit
Remove
Client Service Configuration Detail
Request Generator Configuratio
Details
Integrity
Confidentiality
Security Token
Add
Edit
Remove

Editor for Web Service client security extensions.

The following are port qualified name bindings of the selected service reference.

The following is the request generator configuration for the selected port qualified name binding.

- At the **Security Token** pop-up, perform the following actions:
  - **Name**: **secToken**.
  - **Token type**: Select **Username Token** from the pull-down.
  - Leave **URI** field blank.
  - Ensure that the **Local name** field is automatically populated.
  - Click **OK**.



Name: secToken

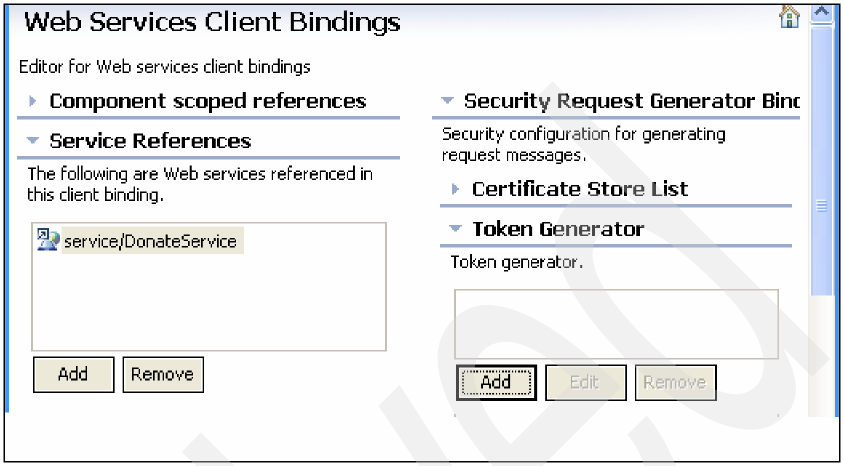
Token type: Username Token

URI:

Local name: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username

2. Add the Token Generator by performing the following actions:
   - At the **Web Deployment Descriptor** editor, switch to the **WS Bindings** tab.
   - On the left, expand **Port Qualified Name Binding**, and select **Donate**.

– On the right, expand **Security Request Generator Binding Configuration** → **Token Generator**, and click the **Add** button located on the right, under the **Token Generator** section.



Web Services Client Bindings

Editor for Web services client bindings

Component scoped references

Service References

The following are Web services referenced in this client binding.

service/DonateService

Security Request Generator Binc

Security configuration for generating request messages.

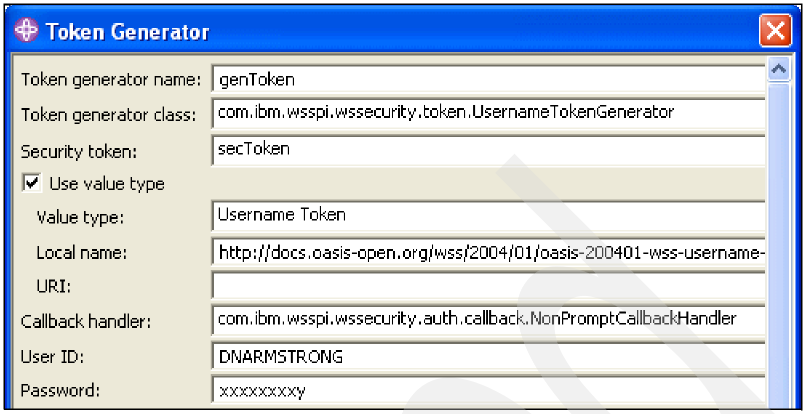
Certificate Store List

Token Generator

Token generator.

– At the **Token Generator** pop-up, perform the following actions:

- **Token generator Name**: **genToken**
- **Token generator class**: Select **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator** from the pull-down.
- **Security token**: Select **secToken** from the pull-down.
- Select **Use value type**.
- **Value type**: Select **Username Token** from the pull-down.
  - **Local Name** should default to **http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken**.
  - **Call back handler** should default to **com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
- **User id**: **DNARMSTRONG**
- **Password**: The password for **DNARMSTRONG**
- Click **OK**.



Token generator name: genToken

Token generator class: com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator

Security token: secToken

☑ Use value type

Value type: Username Token

Local name: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-

URI:

Callback handler: com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler

User ID: DNARMSTRONG

Password: xxxxxxxxy

3. Save and close the **Web Deployment Descriptor**.

Behind the scenes:

The following describes some changes the wizard made:

- **DynamicWeb Projects** → **DonateWSClientWeb** → **WebContent** → **WEB-INF** → **ibm-webservicesclient-ext.xmi** was updated to define that the Web service client should send a security token (secToken) as type Username.
- **DynamicWeb Projects** → **DonateWSClientWeb** → **WebContent** → **WEB-INF** → **ibm-webservicesclient-bnd.xmi** was updated to define genToken, which implements the Username token defined above, including the user ID DNARMSTRONG and an encrypted password.

Note that these definitions are outside the WSDL file. Users of this WSDL file must separately configure the needed WS-Security elements. Therefore, automated tools, such as the Web Services Explorer, that automatically generate proxies solely based on the WSDL do not function after WS-Security is configured on the Web service.

14.5 Update the Web service to support secure access

In this section you update the Web services deployment descriptors to require a username and password.

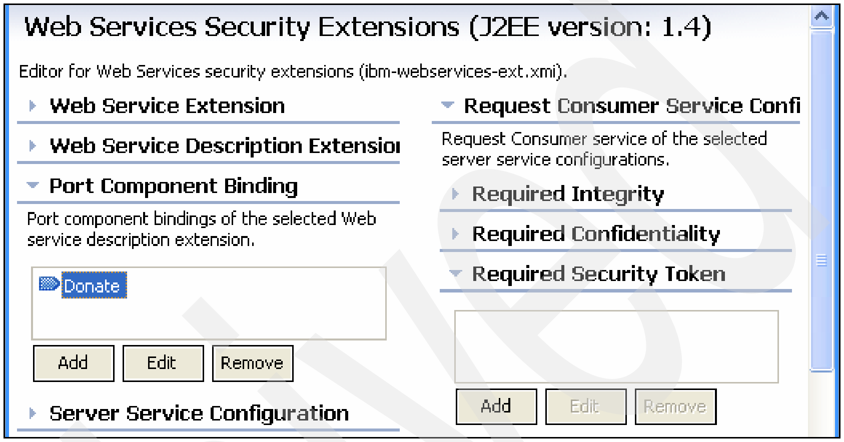
1. From the **Project Explorer**, select **Web Services** → **Services** → **DonateService**, and double-click to open the **Web Services Editor**.

2. Add the Required Security Token by performing the following actions:

   – At the resulting **Web Services Editor,** switch to the **Extensions** tab.

   – Expand **Port Component Binding**, and select **Donate**.

   – Expand **Request Consumer Service Configuration Details** → **Required Security Token**, and click the **Add** button located on the right, under the **Required Security Token** section.



Web Services Security Extensions (J2EE version: 1.4)

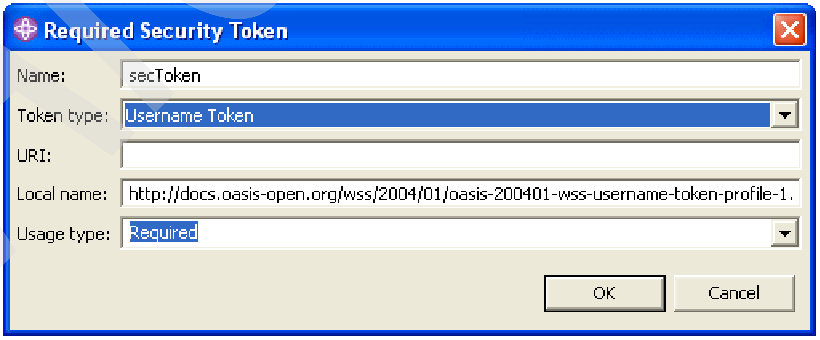
Editor for Web Services security extensions (ibm-webservices-ext.xmi).

Port component bindings of the selected Web service description extension.

Request Consumer service of the selected server service configurations.

   – At the **Required Security Token** pop-up, perform the following actions:

     - **Name**: **secToken**
     - **Token type**: Select **Username** from the pull-down
     - Leave **URI** blank.
     - Ensure that the **Local name** field is automatically populated.
     - Ensure that **Usage type** defaults to **Required**.
     - Click **OK**.



Required Security Token

Name: secToken

Token type: Username Token

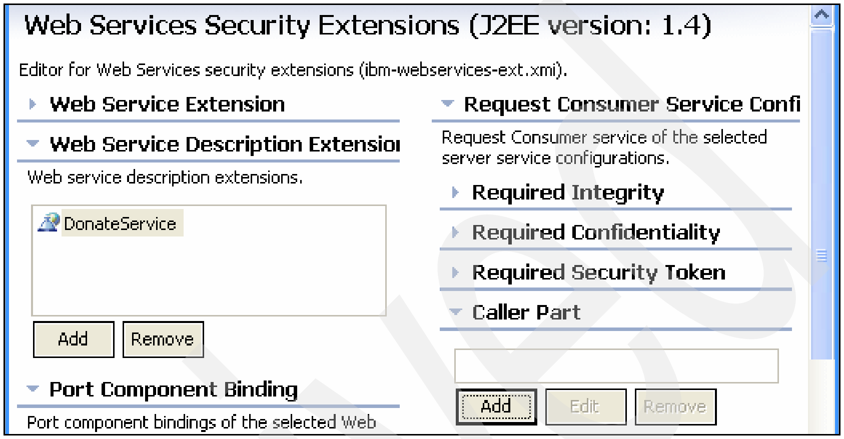
URI:

Local name: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.

Usage type: Required

3. Add the Caller Part by performing the following actions:

- Remain on the **Extensions** tab.
- Expand **Port Component Binding**, and select **Donate**.
- Expand **Request Consumer Service Configuration Details** → **Caller Part**, and click the **Add** button located on the right, under the **Caller Part section**.

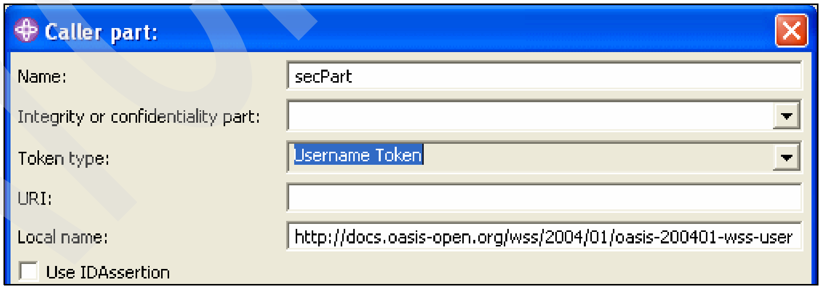


Web Services Security Extensions (J2EE version: 1.4)

Editor for Web Services security extensions (ibm-webservices-ext.xmi).

Web Service Extension
Web Service Description Extensioi
Web service description extensions.
DonateService
Add
Remove
Port Component Binding
Port component bindings of the selected Web
Request Consumer Service Confi
Request Consumer service of the selected server service configurations.
Required Integrity
Required Confidentiality
Required Security Token
Caller Part
Add
Edit
Remove

- At the **Caller Part** pop-up, perform the following actions:
  - **Name**: **secPart**
  - **Token type**: Select **Username Token** from the pull-down (the pull-down is available even though the selection box is grayed out).
  - Ensure that the **Local name** field is automatically populated.
  - Click **OK**.



Caller part:

Name: secPart

Integrity or confidentiality part:

Token type: Username Token

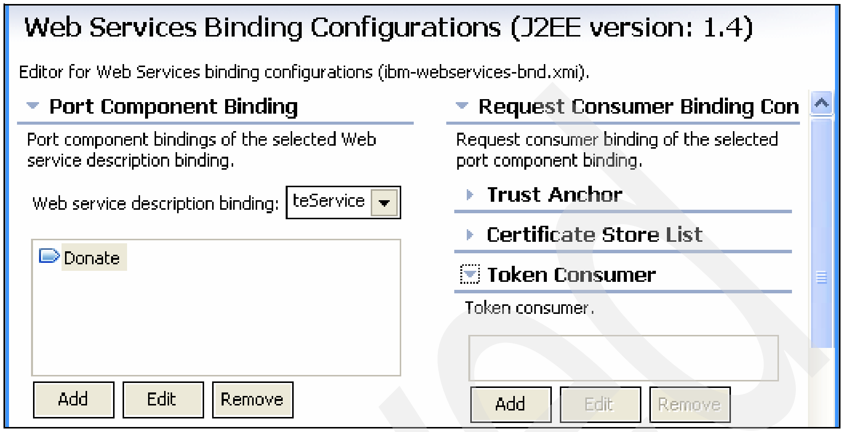
URI:

Local name: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-user

□ Use IDAssertion

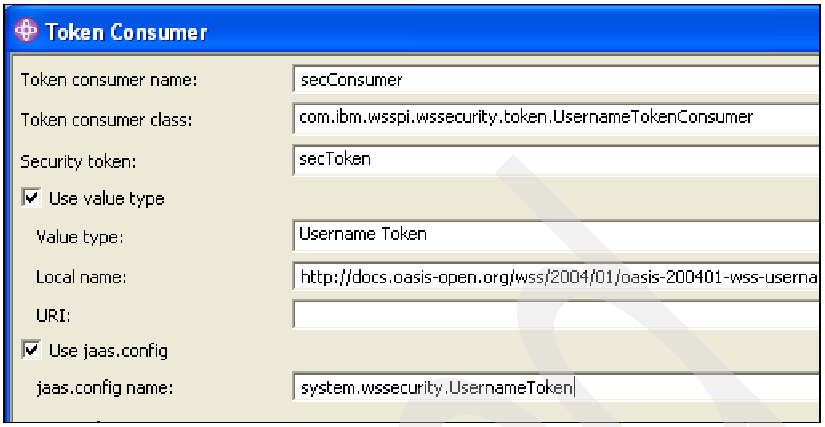
4. Add the Token Consumer by performing the following actions:
   - Switch to the **Binding Configurations** tab.
   - On the left, expand **Port Component Binding** and select **Donate**.

– Expand **Request Consumer Binding Configuration Details** → **Token Consumer**, and click the **Add** button located on the right, under the **Token Consumer** section.


Web Services Binding Configurations (J2EE version: 1.4)
Editor for Web Services binding configurations (ibm-webservices-bnd.xmi).
Port Component Binding
Port component bindings of the selected Web service description binding.
Web service description binding:
teService
Donate
Add
Edit
Remove
Request Consumer Binding Con
Request consumer binding of the selected port component binding.
Trust Anchor
Certificate Store List
Token Consumer
Token consumer.
Add
Edit
Remove

– At the **Token Consumer** pop-up, perform the following actions:

- **Token consumer name**: **secConsumer**
- **Token consumer class** should default to **com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer**.
- **Security Token**: Select **secToken** from the pull-down list.
- Select **User Value type**:
  - **Value type** defaults to **Username Token**.
  - **Local name** defaults to **http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken**.
- Select **Use jaas.config**:
  - **jaas.config name**: **system.wssecurity.UsernameToken**
- Click **OK**.



Token consumer name: secConsumer

Token consumer class: com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer

Security token: secToken

☑ Use value type

Value type: Username Token

Local name: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-userna

URI:

☑ Use jaas.config

jaas.config name: system.wssecurity.UsernameToken

5. Save and close the **Web Services Editor**.

Behind the scenes:

The following describes some changes that the wizard made:

- **EJB Projects** → **DonateEJB2** → **ejbModule** → **META-INF** → **ibm-webservices-ext.xmi** was updated to contain the following:
  - The secPart Caller Part that defines that security should use a username token, which comes from secToken.
- **EJB Projects** → **DonateEJB2** → **ejbModule** → **META-INF** → **ibm-webservices-bnd.xmi** was updated to contain the following:
  - The secConsumer TokenConsumer that receives and validates the secToken security token.

As a result, the call to the Donate session EJB is performed using the identity of the user contained in the username token contained in the inbound SOAP message.

14.6 Test the secure Web service

In this section you test the updated Web service client and Web service.

1. From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWSClientWeb** → **WebContent** → **sampleDonateProxy** → **TestClient.jsp**, and right-click **Run As** → **Run on Server**.

- If you receive the **Define a New Server** pop-up, perform the following actions:
  - Select **Choose an existing server**.
  - Under **Select the server that you want to use**, select **localhost** → **ExperienceJ2EE Server @ localhost**.
  - Optionally select **Set server as project default (do not ask again)**.
  - Click **Finish**.

2. At the resulting **Web Services Test Client** browser, perform the following action:
   - On the left, in the **Methods** pane, select **donateVacation**.
   - On the upper right, in the **Inputs** pane, perform the following actions:
     - **employee_num**: **1**
     - **fund_name**: **DonationFund**
     - **donation_amount**: **1**
     - Click **Invoke**.
   - Note the following successful message should appear in the **Result** pane (lower right):

```
Transfer Successful
```

Off course?

If the successful message does not appear, review the message as well as any messages in the ExperienceJ2EE Server @ localhost log file in the **Console** view for any information regarding the cause.

For example, if you typed the password wrong when entering the information in section "14.4 Update the Web service client to support secure access" on page 336 you would see the following message in the **Result** pane:

```
exception: com.ibm.wsspi.wssecurity.SoapSecurityException:
WSEC6521E: Login failed. The exception is :
javax.security.auth.login.LoginException: WSEC6690E: Failed to
check username [DNARMSTRONG] and password in the UserRegsitry
[sic]: UserRegistryProcessor.checkRegistry()=false
```

You would see a similar message in the **Console** view:

```
/20/06 15:36:17:107 CDT] 00000035 exception     W
com.ibm.ws.wim.adapter.file.was.FileAdapter login CWWIM4512E
The password match failed.
```

3. Close the **Services Test Client** browser.

14.7 Disable J2EE application security

The next scenario in this book, Chapter 15, “Create the message-driven bean” on page 349, requires that the Test Server run without security, so in this section you disable the J2EE application security that you enabled in section “14.2 Re-enable J2EE application security” on page 334.

Disable J2EE application security using the same steps described in section “11.7 Disable J2EE application security” on page 296:

- “11.7.1 Disable J2EE application security”.
- “11.7.2 Restart the ExperienceJ2EE Server @ localhost test server”.

14.8 Explore!

There are no unique explore resources for this chapter. Instead, refer to the resources listed in section “13.6 Explore!” on page 329.

Part 4

Messaging

In Part 4 you extend the core J2EE application to support JMS messaging, by performing the following tasks:

- Chapter 15, “Create the message-driven bean” on page 349: 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 16, “Add publication of results” on page 387: 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 17, “Implement security for messaging” on page 405: Update the messaging artifacts to support security.

15

Create the message-driven bean

In this section 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.

15.1 Learn!

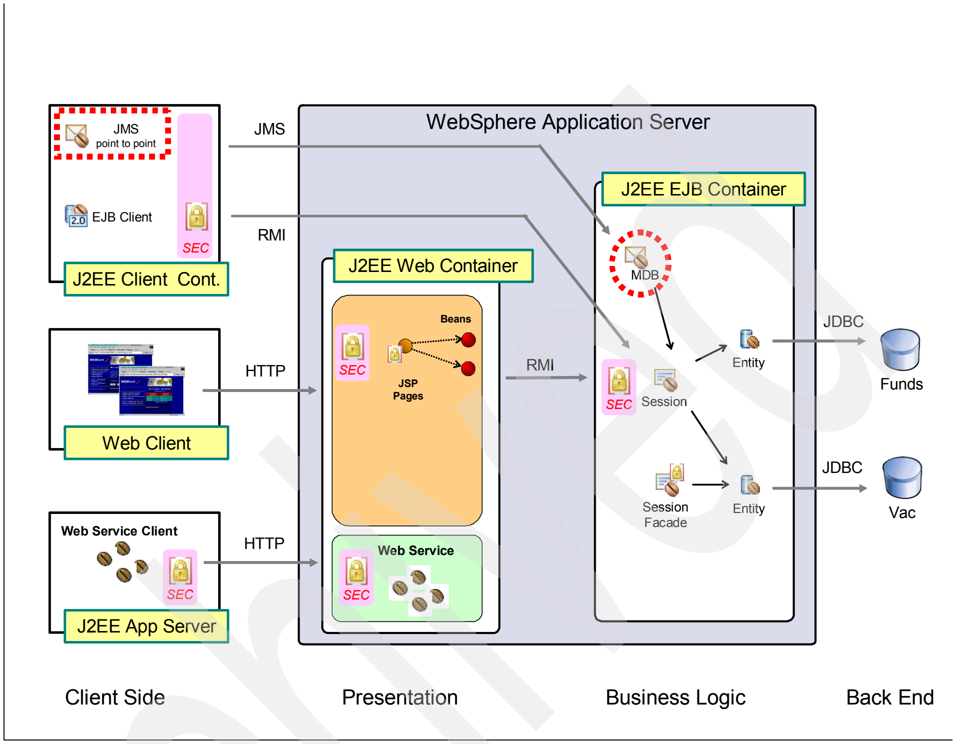


Figure 15-1 Donate Application: Message-driven bean

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 may optionally reply—sending a similar asynchronous message back to the requester.

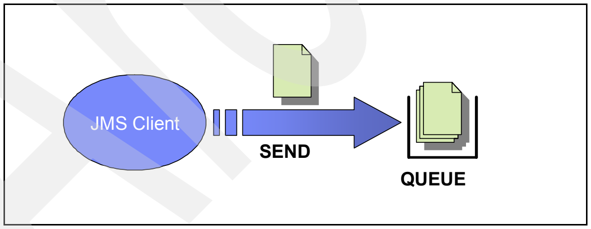
This allows for a loosely coupled solution—applications only have to be aware that the message was ‘accepted’ for delivery and then can proceed to other processing. An outage in any one part does not immediately block operation in the rest of the system.

J2EE 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 IBM WebSphere MQ or the Messaging engine component used here).

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 J2EE server so far was over HTTP. How do you access J2EE over messaging?

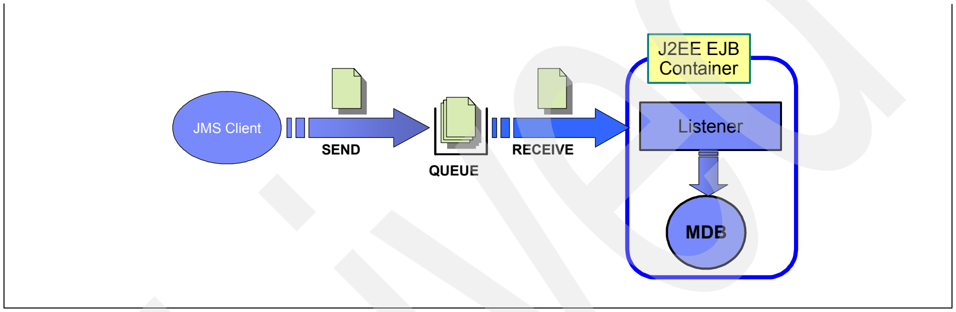
- **Client side**: A program creates a message and sends it to a JMS queue, in this case a local queue—the message stays on the source system and is not forwarded to a remote system, and then terminates. The program does not know when (or if) the message is actually read, only that the message is persisted in the queue.



  In this scenario the program is a standalone Java program, but you could put the same basic code inside J2EE server side artifacts (such as servlets or EJBs) if you wanted to send a message from one J2EE server to another, or even to another component in the same J2EE server.

- **J2EE 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 deployed to this EJB container, referencing the same activation specification. 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 J2EE container. The MDB is invoked by the overall J2EE infrastructure and does not have any external home or local interfaces.



The following additional learn resources are available:

- J2EE: Java Message Service API – FAQ:

  `http://java.sun.com/products/jms/faq.html`

- J2EE: The J2EE 1.4 Tutorial, Chapters 33 (The Java Message Service API) and 34 (J2EE Examples Using the JMS API):

  `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`

- IBM developerWorks: J2EE pathfinder: Enterprise messaging with JMS:

  `http://www.ibm.com/developerworks/java/library/j-pj2ee5/index.html`

- WAS V6.1 InfoCenter: Service integration buses:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.pmc.doc/tasks/tjj9999_.html`

15.1.1 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:

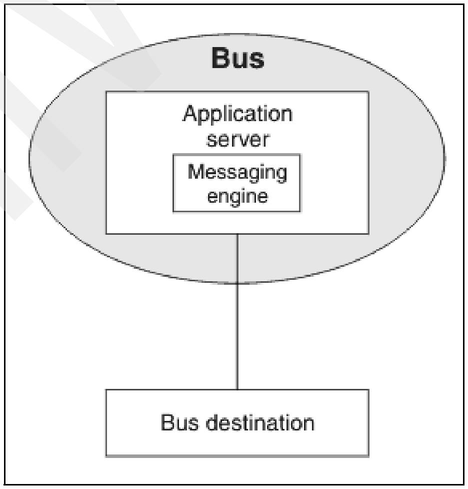
- Messages can be sent and received over multiple protocols—base JMS, Web services over HTTP, and Web services over JMS. Messages can be received in one protocol, and resent in another.

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.

15.2 Configure the test server for base JMS resources

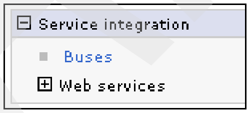
In this section you configure the **ExperienceJ2EE Server @ localhost** test server with the required resources for the MDB:

- Create the base SIB definition in section 15.2.1, “Create the Service Integration Bus” on page 354.
- Configure security in the SIB in section 15.2.2, “Configure security” on page 358
- Create the Donate queue in the SIB in section 15.2.3, “Create the destination queue resource” on page 360.
- Create the JMS queue definition that provides the link to the Donate queue in the SIB in section 15.2.4, “Create the JMS queue destination” on page 363.
- Create the JMS activation specification (needed for the MDB) in section 15.2.5, “Create the JMS activation specification” on page 365.
- Create the JMS queue connection factory in section 15.2.6, “Create the JMS connection factory for the application client” on page 367.

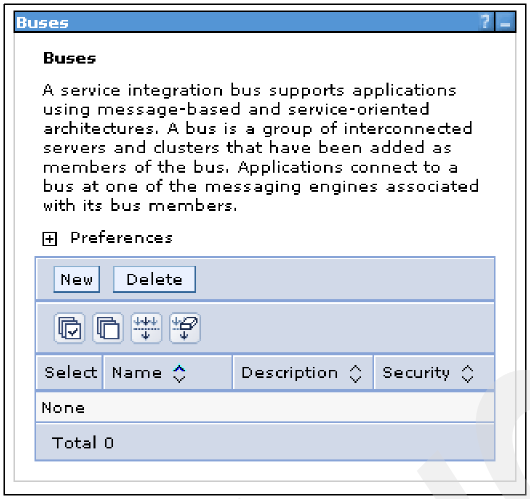
  This resource is not used by the MDB but instead is used by the J2EE application client defined later in this chapter and the pub sub logic added in the subsequent chapter. It is defined here to keep all JMS related definitions in one section.

15.2.1 Create the Service Integration Bus

1. Start the **Admin Console** using the same steps as described in section “5.4.1 Start the admin console” on page 102.
   – Recall that at the **Admin Console** login screen you use the **j2eeadmin** user ID and password that you defined in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51.
2. From the Admin Console, on the left, expand to **Service integration**, and click **Buses**.



3. In the **Buses** screen, click **New**.



Buses

Buses

A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

⊞ Preferences

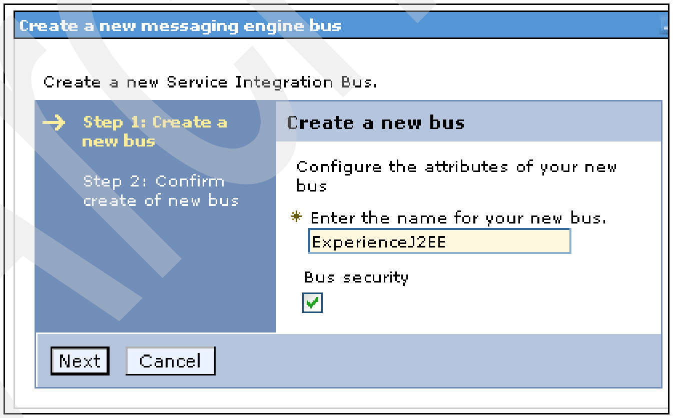
Select	Name	Description	Security
None			

Total 0

– On the right, in the **Create a new messaging engine bus** screen, perform the following actions:

- **Enter the name for your new bus**: **ExperienceJ2EE**
- Leave **Bus security** selected.
- Click **Next**.

– On the right, in the **Create a new messaging engine bus** screen, click **Finish**.



Create a new messaging engine bus

Create a new Service Integration Bus.

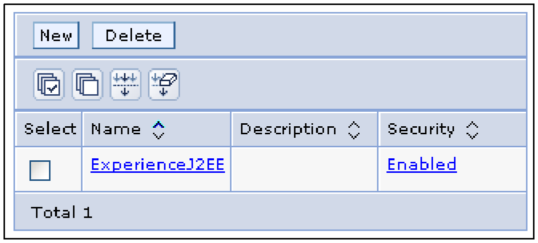
Create a new bus

Configure the attributes of your new bus

* Enter the name for your new bus.
ExperienceJ2EE

Bus security
☑

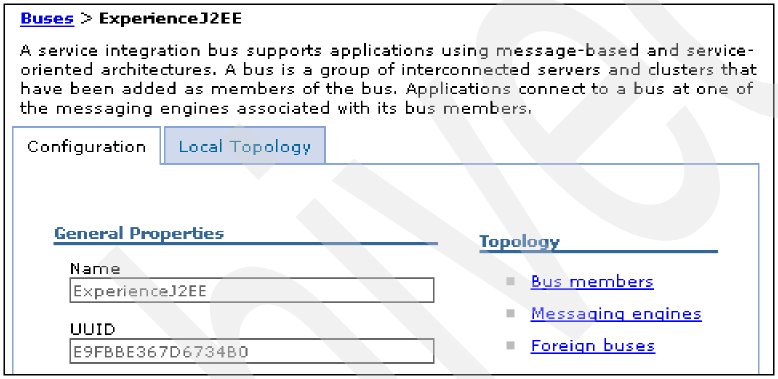
4. On the **Buses** screen, which is updated with the new entry, scroll down, and click the **ExperienceJ2EE** bus.



Select	Name	Description	Security
□	ExperienceJ2EE		Enabled

Total 1

5. On the right, in the **Buses** → **ExperienceJ2EE** screen, under **Topology**, click **Bus members**.


Buses > ExperienceJ2EE

A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

General Properties

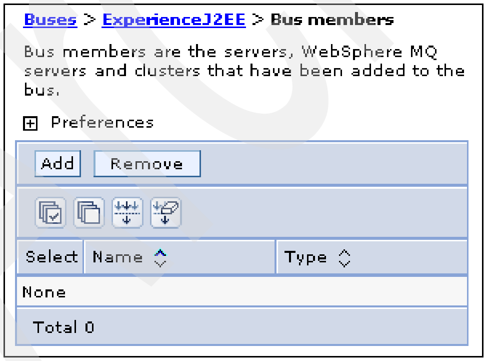
Name
ExperienceJ2EE

UUID
E9FBBE367D6734B0

Topology

- Bus members
- Messaging engines
- Foreign buses

6. On the right, at the **Buses** → **ExperienceJ2EE** → **Bus members** screen, click **Add**.

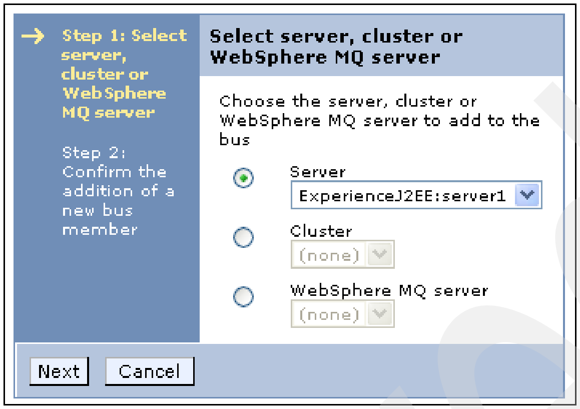

Buses > ExperienceJ2EE > Bus members

Bus members are the servers, WebSphere MQ servers and clusters that have been added to the bus.

Select	Name	Type
None		

Total 0

- At the **Step 1: Select server, cluster, or WebSphere MQ Server** screen, ensure that the **Server** option is enabled and that the server is **ExperienceJ2EE:server1**, and click **Next**.



→ **Step 1: Select server, cluster or WebSphere MQ server**

Step 2: Confirm the addition of a new bus member

Select server, cluster or WebSphere MQ server

Choose the server, cluster or WebSphere MQ server to add to the bus

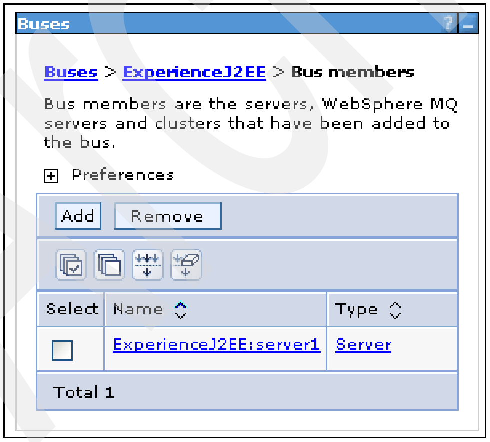
● Server: ExperienceJ2EE:server1

○ Cluster: (none)

○ WebSphere MQ server: (none)

- At the **Step 2: Select the type of message store** screen, accept the default selection of **File store**, and click **Next**.
- At the **Step 3: Provide the message store properties** screen, accept the default selections, and click **Next**.
- At the **Step 4: Confirm the addition of a new bus member** screen, click **Finish**.

7. At **the Buses** → **ExperienceJ2EE** → **Bus Members** screen, verify that bus member was added as the following graphic illustrates:



Buses

Buses > ExperienceJ2EE > **Bus members**

Bus members are the servers, WebSphere MQ servers and clusters that have been added to the bus.

⊞ Preferences

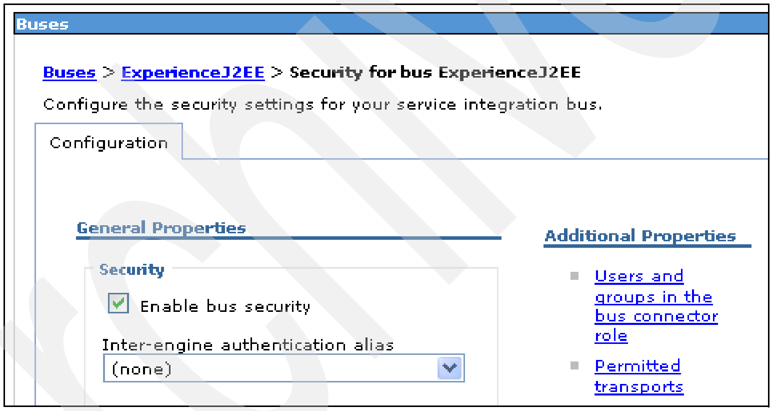
Select	Name	Type
□	ExperienceJ2EE:server1	Server
Total 1		

15.2.2 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 J2EE application security) is required to connect to the bus in order to send and receive messages.

For simplicity, configure the bus to allow access by any authenticated user (similar to the ALLAUTHENTICATED_ROLE you used in section 11.4, “Implement declarative security” on page 266).

1. From the **Admin Console**, on the left expand to **Service integration**, and click **Buses**.
2. On the right, at the **Buses** screen, click **ExperienceJ2EE**.
3. On the right, at the **Buses** → **ExperienceJ2EE** screen, under **Additional properties** click **Security**.
4. On the right at the **Buses** → **ExperienceJ2EE** → **Security for bus ExperienceJ2EE** screen, under **Additional Properties** click **Users and groups in the bus connector role**.



Buses

Buses > ExperienceJ2EE > Security for bus ExperienceJ2EE

Configure the security settings for your service integration bus.

Configuration

General Properties

Security

☑ Enable bus security

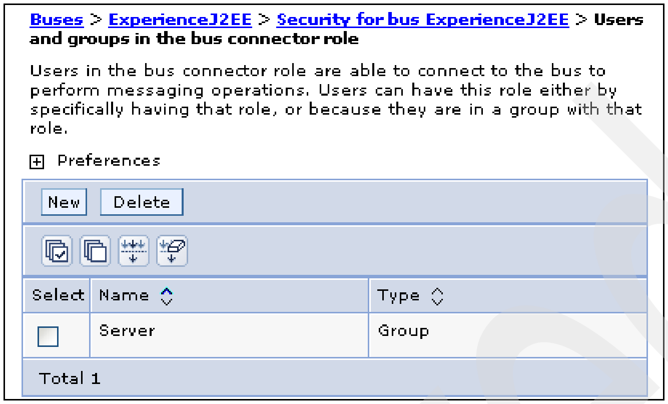
Inter-engine authentication alias

(none)

Additional Properties

- Users and groups in the bus connector role
- Permitted transports

5. On the right, at the **Buses → ExperienceJ2EE → Security for bus ExperienceJ2EE → Users and groups in the bus connector role** screen, click **New**.



Buses > ExperienceJ2EE > Security for bus ExperienceJ2EE > **Users and groups in the bus connector role**

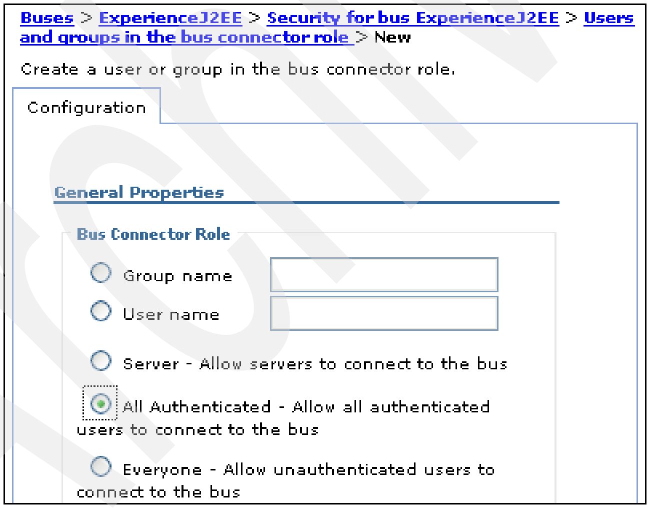
Users in the bus connector role are able to connect to the bus to perform messaging operations. Users can have this role either by specifically having that role, or because they are in a group with that role.

⊞ Preferences

Select	Name	Type
☐	Server	Group

Total 1

– On the right, at the **Buses → ExperienceJ2EE → Security for bus ExperienceJ2EE → Users and groups in the bus connector role → New** screen, select **All Authenticated - Allow all authenticated users to connect to the bus**, and click **OK**.



Buses > ExperienceJ2EE > Security for bus ExperienceJ2EE > Users and groups in the bus connector role > New

Create a user or group in the bus connector role.

General Properties

Bus Connector Role

○ Group name

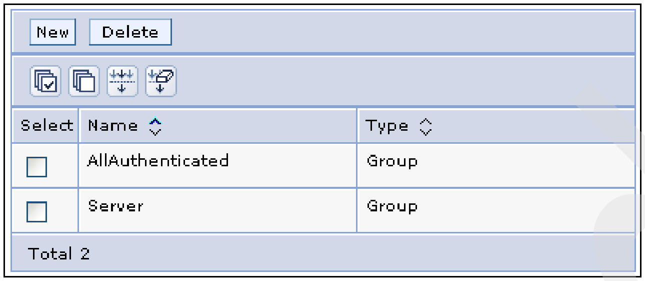
○ User name

○ Server - Allow servers to connect to the bus

● All Authenticated - Allow all authenticated users to connect to the bus

○ Everyone - Allow unauthenticated users to connect to the bus

6. At the **Buses** → **ExperienceJ2EE** → **Security for bus ExperienceJ2EE** → **Users and groups in the bus connector role** screen, verify that the role was added.



Select	Name	Type
□	AllAuthenticated	Group
□	Server	Group
Total 2		

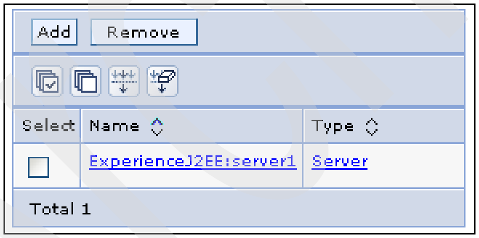
7. At the top of the panel, in the **Messages** box that states "**Changes have been made to your local configuration….**", click **Save**.

15.2.3 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).

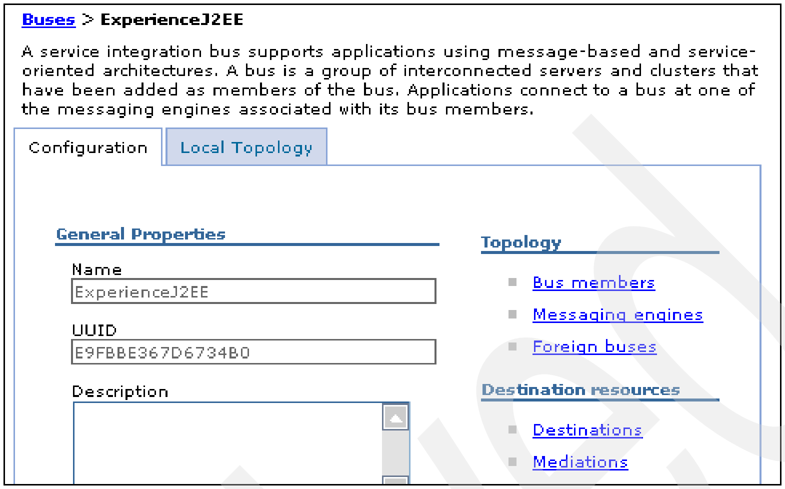
1. From the **Admin Console**, on the left, expand to **Service integration** and click on **Buses**.

2. On the right, at the **Buses** screen, click **ExperienceJ2EE**.



Select	Name	Type
□	ExperienceJ2EE:server1	Server
Total 1		

3. On the right at the **Buses** → **ExperienceJ2EE** screen, under **Destination resources**, click **Destinations**.



Buses > ExperienceJ2EE

A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

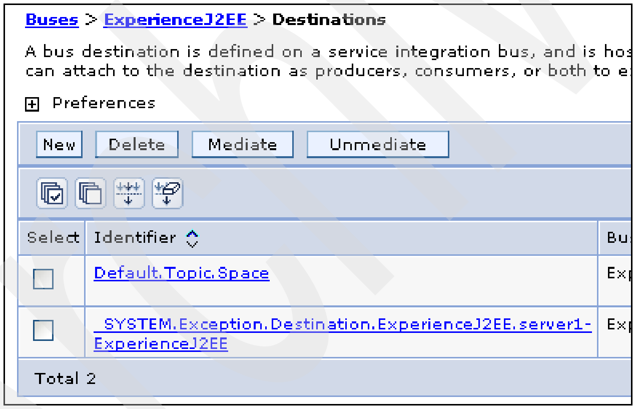
General Properties

Name
ExperienceJ2EE

UUID
E9FBBE367D6734B0

Description

4. On the right at the **Buses** → **ExperienceJ2EE** → **Destinations** screen, click **New**.

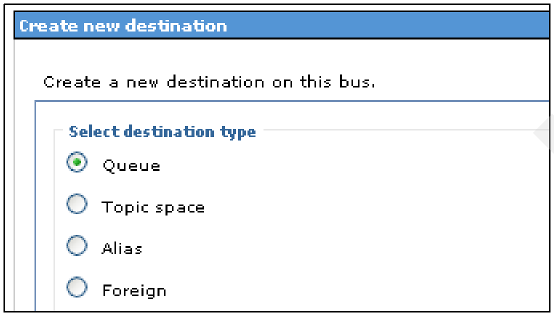


Buses > ExperienceJ2EE > Destinations

A bus destination is defined on a service integration bus, and is hos
can attach to the destination as producers, consumers, or both to e

Select	Identifier ⇕	Bu
□	Default.Topic.Space	Ex
□	_SYSTEM.Exception.Destination.ExperienceJ2EE.server1-ExperienceJ2EE	Ex
Total 2		

- At the **Create a new destination** screen, 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

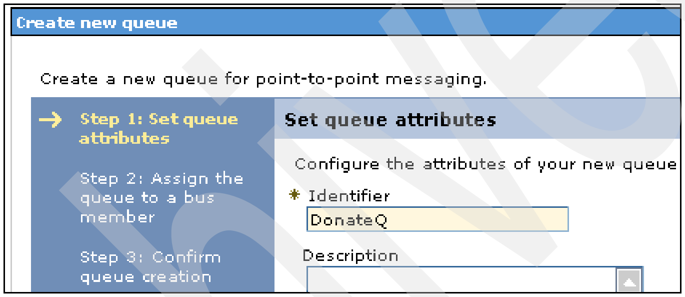
● Queue

○ Topic space

○ Alias

○ Foreign

- At the **Step1: Set queue attributes** screen, 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

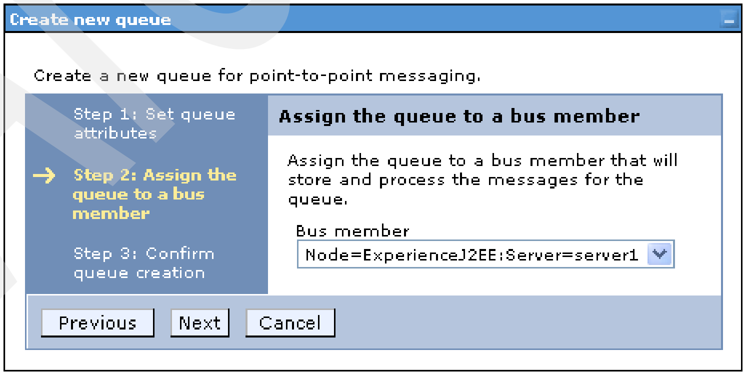
Set queue attributes

Configure the attributes of your new queue

* Identifier: DonateQ

Description

- At the **Step 2: Assign the queue to a bus member** screen, verify that the bus is set to **Node=ExperienceJ2EE:Server1=server1**, and click **Next**.
- At the **Create new queue** (**Step 3: Confirm queue creation)** screen, click **Finish**.



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

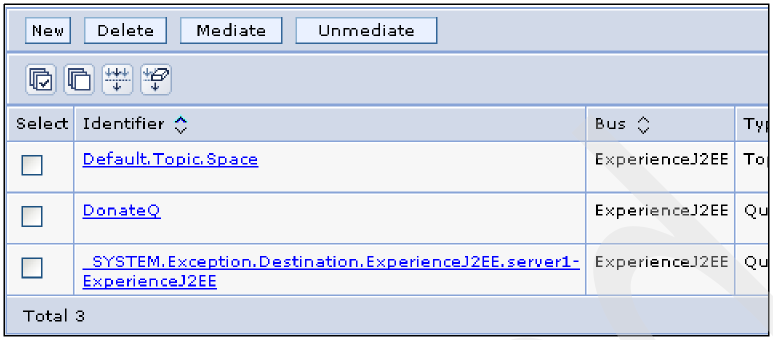
Step 3: Confirm queue creation

Assign the queue to a bus member

Assign the queue to a bus member that will store and process the messages for the queue.

Bus member: Node=ExperienceJ2EE:Server=server1

5. Back at the **Buses** → **ExperienceJ2EE** → **Destinations** screen, verify that the queue destination was added.



Select	Identifier	Bus	Ty
□	Default.Topic.Space	ExperienceJ2EE	To
□	DonateQ	ExperienceJ2EE	Qu
□	_SYSTEM.Exception.Destination.ExperienceJ2EE.server1-ExperienceJ2EE	ExperienceJ2EE	Qu

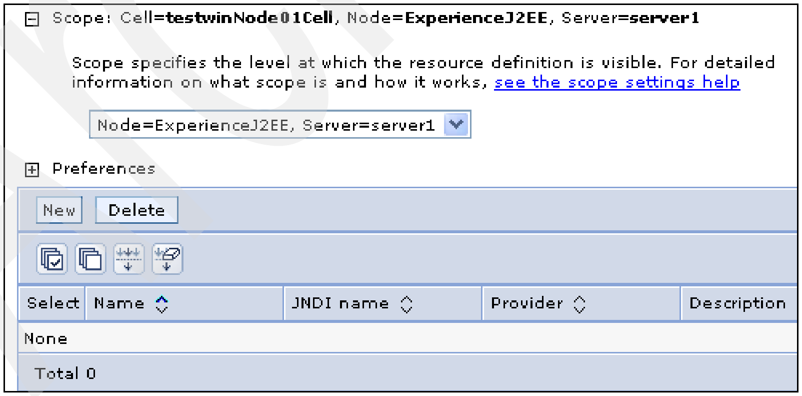
Total 3

6. At the top of the panel, in the **Messages** box that states "**Changes have been made to your local configuration….**", click **Save**.

15.2.4 Create the JMS queue destination

A JMS queue provides the linkage to the physical resource, in this case the DonateQ destination defined in step 4 of the previous section.

1. From the **Admin Console**, on the left expand to **Resources** → **JMS**, and click **Queues**.
2. On the right, at the **Queues** screen, select **Node-ExperienceJ2EE, Server = server1** from the scope pull-down.
3. On the right, at the refreshed **Queues** screen, click **New**.


Scope: Cell=**testwinNode01Cell**, Node=**ExperienceJ2EE**, Server=**server1**

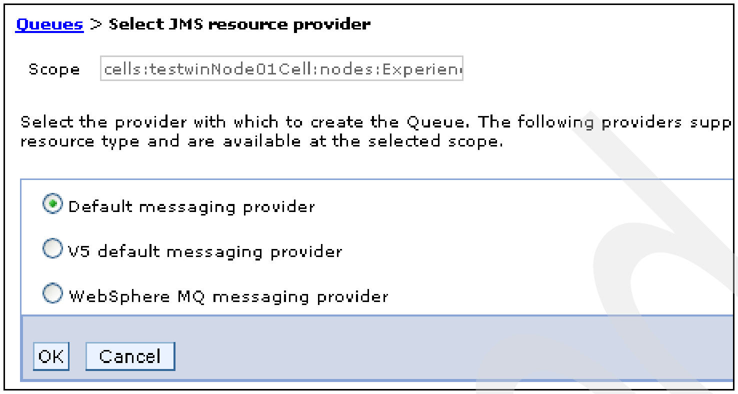
Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help

Node=ExperienceJ2EE, Server=server1

Select	Name	JNDI name	Provider	Description
None				

Total 0

4. On the right, at the **Queues** → **Select JMS resource provider screen**, select **Default messaging provider**, and click **OK**.



Queues > Select JMS resource provider

Scope cells:testwinNode01Cell:nodes:Experien

Select the provider with which to create the Queue. The following providers supp resource type and are available at the selected scope.

● Default messaging provider

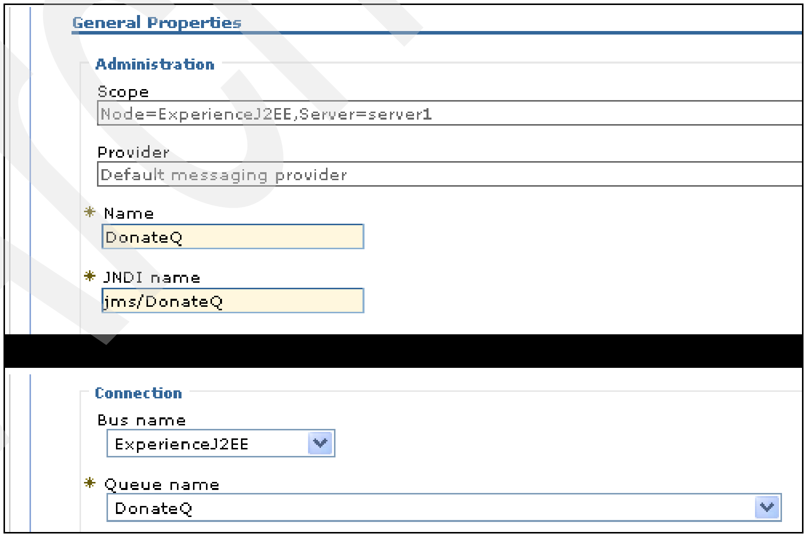
○ V5 default messaging provider

○ WebSphere MQ messaging provider

OK Cancel

5. On the right, at the **Queues** → **Default messaging provider** → **Queues** → **New** screen, perform the following actions. Scroll down to see all the following values and buttons.

- **Name**: **DonateQ**
- **JNDI name**: **jms/DonateQ**
- **Bus Name**: **ExperienceJ2EE**
- **Queue Name**: **DonateQ**
- Click **OK**.



General Properties

Administration

Scope
Node=ExperienceJ2EE,Server=server1

Provider
Default messaging provider

* Name
DonateQ

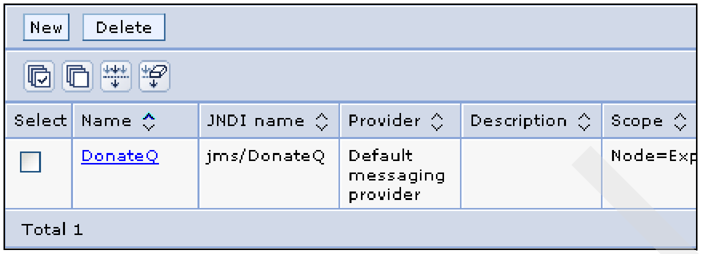
* JNDI name
jms/DonateQ

Connection

Bus name
ExperienceJ2EE

* Queue name
DonateQ

The **Queues** screen should reflect the new entry.



New Delete

Select	Name	JNDI name	Provider	Description	Scope
□	DonateQ	jms/DonateQ	Default messaging provider		Node=Exp

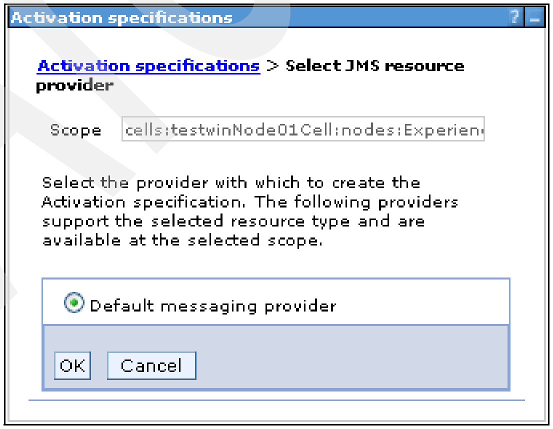
Total 1

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

15.2.5 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. From the **Admin Console**, on the left expand to **Resources** → **JMS**, and click **Activation specifications**.
2. On the right, at the **Activation specifications** screen, select **Node=ExperienceJ2EE, Server = server1** from the scope pull-down.
3. On the right, at the refreshed **Activation specifications** screen, click **New**.
   – On the right, at the **Activation specifications** → **Select JMS resource provider** screen, accept the selection of **Default messaging provider**, and click **OK**.



Activation specifications

Activation specifications > Select JMS resource provider

Scope cells:testwinNode01Cell:nodes:Experien

Select the provider with which to create the Activation specification. The following providers support the selected resource type and are available at the selected scope.

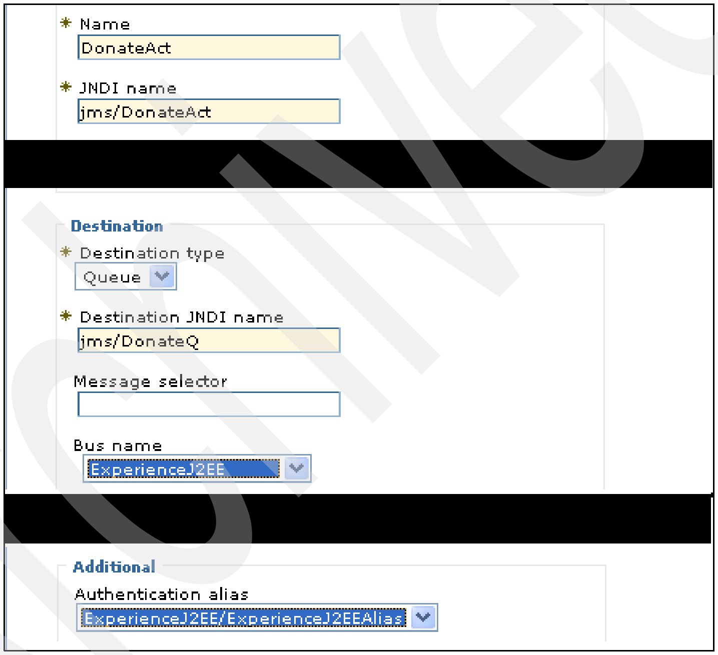
● Default messaging provider

OK Cancel

– On the right at the **Activation specifications** → **Default messaging provider** → **New** screen, perform the following actions. Scroll down to see all of the following values and buttons:

Name:	**DonateAct**
JNDI Name:	**jms/DonateAct**
Destination type:	**Queue**
Destination JNDI Name:	**jms/DonateQ**
Bus name:	**ExperienceJ2EE**
Authentication alias:	**ExperienceJ2EE/ExperienceJ2EEAlias**

- Click **OK**.



* Name
DonateAct

* JNDI name
jms/DonateAct

Destination

* Destination type
Queue

* Destination JNDI name
jms/DonateQ

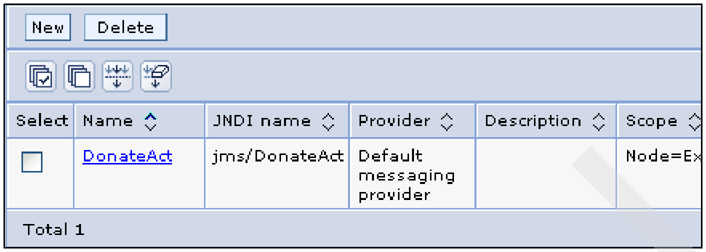
Message selector

Bus name
ExperienceJ2EE

Additional

Authentication alias
ExperienceJ2EE/ExperienceJ2EEAlias

The **Activation specification** screen should reflect the new entry:



Select	Name	JNDI name	Provider	Description	Scope
□	DonateAct	jms/DonateAct	Default messaging provider		Node=Ex

Total 1

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

15.2.6 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 UNAUTHENICATED 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 section 15.4, “Create the messaging client (DonateMDBClient)” on page 377.

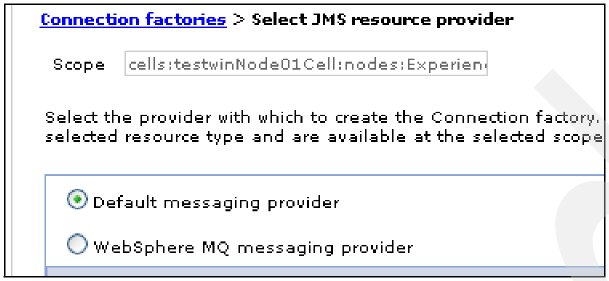
Note that this connection factory is NOT required for the MDB that you create in section 15.3, “Create the message-driven bean” on page 370. That artifact accesses the class of service through the activation specification.

Thus, you can defer this configuration step until section 15.4, “Create the messaging client (DonateMDBClient)” on page 377, but it is done in this section to keep all the JMS configuration activities in one location.

1. From the **Admin Console**, on the left expand to **Resources** → **JMS**, and click **Connection factories**.
2. On the right, at the **Connection factories** screen, select **Node=ExperienceJ2EE, Server = server1** from the scope pull-down.

3. On the right, at the refreshed **Connection factories** screen, click **New**.

   – At the **Connection factories** → **Select JMS resource provider screen**, accept the selection of **Default messaging provider**, and click **OK**.



Connection factories > Select JMS resource provider

Scope: cells:testwinNode01Cell:nodes:Experien

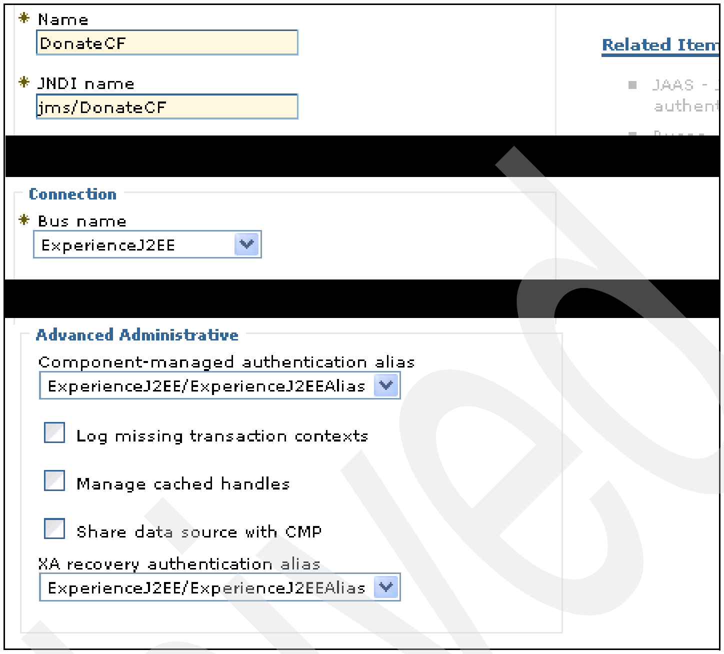
Select the provider with which to create the Connection factory. selected resource type and are available at the selected scope

● Default messaging provider

○ WebSphere MQ messaging provider

   – On the right, at the **Connection factories** → **Default messaging provider** → **New screen**, perform the following actions. Scroll down the screen to see all the following selections:

     - **Name**: **Donate CF**
     - **JNDI name**: **jms/DonateCF**
     - **Bus Name**: **ExperienceJ2EE**
     - **Component-managed authentication alias**: **ExperienceJ2EE/ExperienceJ2EEAlias**
     - **XA recovery authentication alias**: **ExperienceJ2EE/ExperienceJ2EEAlias**
     - Click **OK**.



* Name
DonateCF

* JNDI name
jms/DonateCF

[redacted]

Connection

* Bus name
ExperienceJ2EE

[redacted]

Advanced Administrative

Component-managed authentication alias
ExperienceJ2EE/ExperienceJ2EEAlias

□ Log missing transaction contexts

□ Manage cached handles

□ Share data source with CMP

XA recovery authentication alias
ExperienceJ2EE/ExperienceJ2EEAlias

Behind the scenes:

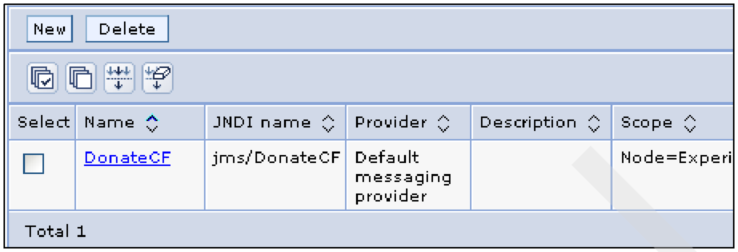
J2EE application clients cannot use the JAAS authentication alias information that is specified in the connection factory. Instead, they must supply this information using the createConnection method (as you do in section 15.4, “Create the messaging client (DonateMDBClient)” on page 377).

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 16, “Add publication of results” on page 387, 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.

The resulting **Connection factories** screen should reflect the new entry.



Select	Name	JNDI name	Provider	Description	Scope
□	DonateCF	jms/DonateCF	Default messaging provider		Node=Experi

Total 1

4. At the top of the panel, in the **Messages** box that states "**Changes have been made to your local configuration….**", click **Save**.
5. Logout and close the **Admin Console**.

15.2.7 Restart the ExperienceJ2EE Server @ localhost test server

Restart the test server because of the activation of the service integration bus and messaging engine.

1. Switch to the **Servers** view in the lower-right pane, select **ExperienceJ2EE Server @ localhost**, and select the Restart icon [ ] in the action bar. 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
```

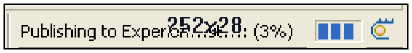
15.3 Create the message-driven bean

In this section you use the standard AST V6.1 EJB tooling to create the message-driven bean (MDB).

15.3.1 Work around: Disable automatic publishing

After creating an MDB for a project that is deployed to a test server, AST V6.1 (both base code and with Fixpack 1) enters an endless publishing loop when making updates.

The background task status bar in the lower right constantly loops from 0% to 100%, as the following graphic illustrates:



Publishing to Experien...se...: (3%)

And the **Console** view shows the following two repeating messages (among many others):

```
FileRepositor A   ADMR0015I: User defaultWIMFileBasedRealm/j2eeadmin
created document
cells/testwinNode01Cell/applications/DonateEAR.ear/deltas/DonateEAR/
delta-1157748392515.
FileRepositor A   ADMR0016I: User defaultWIMFileBasedRealm/j2eeadmin
modified document
cells/testwinNode01Cell/applications/DonateEAR.ear/deployments/Donat
eEAR/deployment.xml.
```

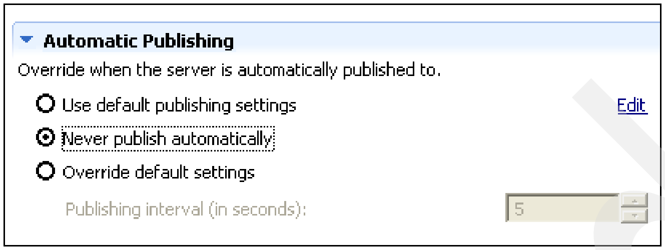
If you open delta-xxxxxxxxxxxxx, observe that ibm_ejbext.properties was identified as being updated. And if you look at the details of this on the file system you observe that is indeed being updated each time the application is published:

```
<?xml version="1.0" encoding="UTF-8"?>
<app-delta>
<change_input changetype="fg-update" contenttype="partialapp"
contenturi="C:/Workspace/ExperienceJ2EE_test/.metadata/.plugins/org.
eclipse.wst.server.core/tmp1/DonateEAR.ear"/>
<files>
<file operation="update"
uri="/DonateEJB2.jar/META-INF/ibm_ejbext.properties"/>
</files>
</app-delta>
```

This is a known defect with AST V6.1, and the availability date for a fix is to be determined.

1. Switch to the **Servers** view in the lower right.
2. Select **ExperienceJ2EE v6.1 Server @ localhost**, and double-click to open.

3. In the resulting **ExperienceJ2EE Server @ localhost** editor, in the upper-center pane, expand the **Automatic Publishing** section located in the upper right, and select **Never publish automatically**.



Automatic Publishing

Override when the server is automatically published to.

○ Use default publishing settings   Edit

● Never publish automatically

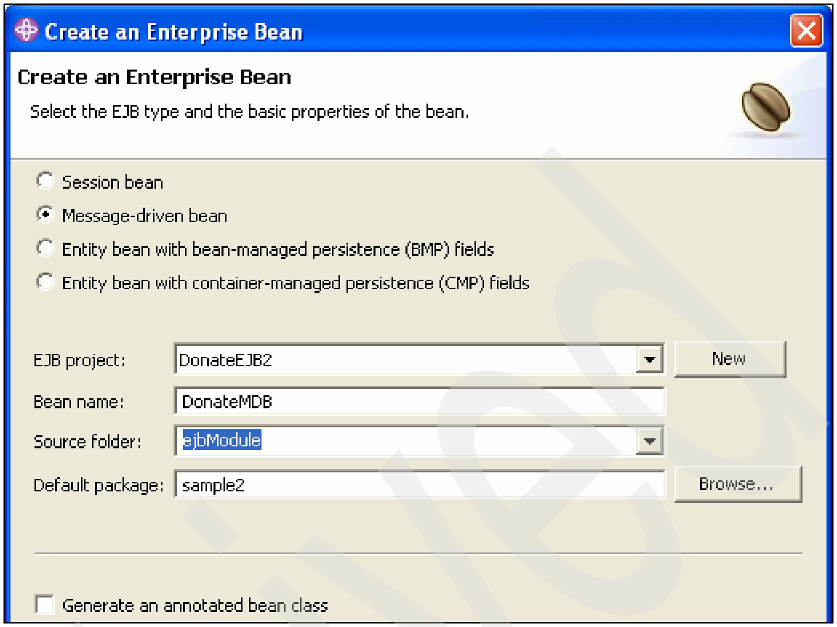
○ Override default settings

Publishing interval (in seconds): 5

4. Save and close the editor.

15.3.2 Define the MDB

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Message-Driven Beans**, and right-click **New** → **Message-Driven Bean**.
2. At the **Create an Enterprise Bean** pop-up, perform the following actions:
   - Ensure that **Message-driven bean** is selected.
   - Ensure that **EJB Project** is set to **DonateEJB2**.
   - **Bean name**: **DonateMDB**
   - Ensure that **Source folder** is set to **ejbModule**.
   - **Default package**: **sample2**
   - Clear **Generate an annotated bean class**.
   - Click **Next**.



Create an Enterprise Bean

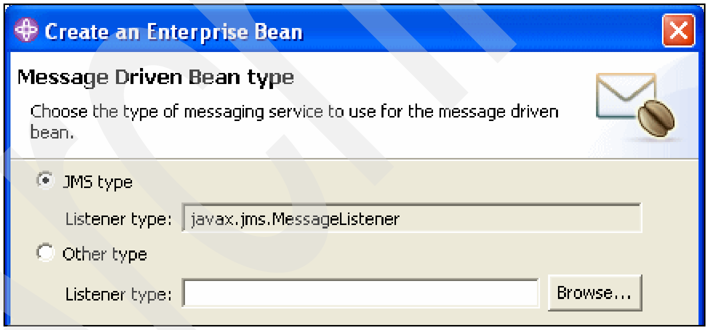
Select the EJB type and the basic properties of the bean.

○ Session bean
● Message-driven bean
○ Entity bean with bean-managed persistence (BMP) fields
○ Entity bean with container-managed persistence (CMP) fields

EJB project: DonateEJB2
Bean name: DonateMDB
Source folder: ejbModule
Default package: sample2

□ Generate an annotated bean class

3. At the **Message Driven Bean type** pop-up, accept the default selection of JMS type / javax.jms.MessageListener, and click **Finish**.



Message Driven Bean type

Choose the type of messaging service to use for the message driven bean.

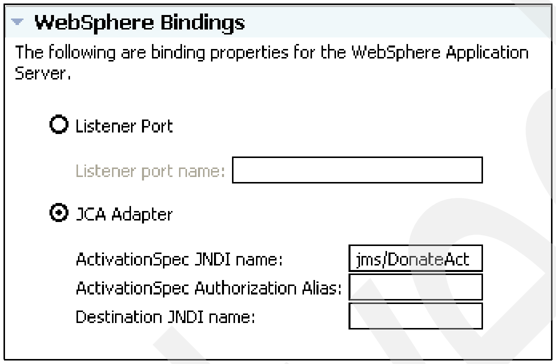
● JMS type
Listener type: javax.jms.MessageListener
○ Other type
Listener type:

15.3.3 Add the activation specification information

In this section you update the MDB configuration to reference the same activation specification that you defined earlier in section 15.2.5, “Create the JMS activation specification” on page 365.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2**, and double-click to open.
2. At the **EJB Deployment Descriptor** editor, switch to the **Bean** tab.

- On the left, select **DonateMDB**.
- On the lower right (below the gray line), expand **WebSphere Bindings** and perform the following actions:
  - Select **JCA Adapter.**
  - **ActivationSpec JNDI name**: **jms/DonateAct**



▾ WebSphere Bindings

The following are binding properties for the WebSphere Application Server.

○ Listener Port

Listener port name:

● JCA Adapter

ActivationSpec JNDI name: jms/DonateAct

ActivationSpec Authorization Alias:

Destination JNDI name:

3. Save and close the **EJB Deployment Descriptor**.

15.3.4 Create the EJB reference

The MDB accesses the Donate session EJB. Per J2EE best practices, you access this by using an EJB reference.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2**, and double-click to open the deployment descriptor.
2. From the resulting **EJB Deployment Descriptor** editor, switch to the **References** tab.
3. Add the Donate EJB reference by performing the following actions:
   - Under References, select **DonateMDB**, and click **Add...**
   - At the resulting **Reference** pop-up, select **EJB reference**, and click **Next**.
   - At the **EJB Reference** pop-up, perform the following actions:
     - Select **Enterprise Beans in the workspace**.
     - Select **DonateEAR** → **DonateEJB2** → **Donate**. Note that the **Name** field is automatically populated with **ejb/Donate**.
     - Set **Ref Type** to **Remote**.
   - Click **Finish**.
4. Save and close the EJB deployment descriptor.

15.3.5 Code the DonateMDB message-driven bean

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Message-Driven Beans** → **DonateMDB** → **DonateMDBBean**(.java), and double-click to open the Java class.
2. From the resulting **DonateMDBBean.java** editor, insert the handleMessage method using the following actions:
   - In the **DonateMDBean.java** editor, insert a blank line before the closing brace, and position the cursor at this new line.
   - In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG10: DonateMDB handleMessage**.
   - Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).
   - At the **Organize Imports** pop-up, select **java.util.StringTokenizer**, and click **Finish**. This removes all errors except one (`aDonate cannot be resolved`).

```
private void handleMessage(javax.jms.Message msg) {
   try {
      String text = ((TextMessage) msg).getText();
      System.out.println("DonateMDB: Input Message = " + text);
      StringTokenizer tokens = new StringTokenizer(text, " ", false);
      int employee_num = Integer.parseInt(tokens.nextToken());
      String fund_name = tokens.nextToken();
      int donation_amount = Integer.parseInt(tokens.nextToken());
      try {
   //* Insert call to EJB Create Method

         String resultMessage = aDonate.donateVacation(employee_num,
fund_name, donation_amount);
         System.out.println("DonateMDB: call to donateVacation
returned  " + resultMessage);
      } catch (Exception e) {
         System.out.println("DonateMDB: error on call to
donateVacation:" + e.getLocalizedMessage());
      }
   } catch (Exception e) {
      System.out.println("DonateMDB: error on parsing message: " +
e.getLocalizedMessage());
   }
}
```

Figure 15-2 FRAG10: DonateMDB handleMessage

3. Insert the call to the Donate session EJB by performing the following actions:
   - Select the line after **//* Insert call to EJB Create Method**.
   - In the **Snippets** view (lower right), expand the **EJB** category, and double-click **Call an EJB "create" method**.
   - At the **Select Reference** pop-up, select **ejb/Donate**, and click **Finish.**

     Note that the following line was inserted into **DonateMDBBean.java**:

     ```
     Donate aDonate = createDonate();
     ```

     If you save the file, all errors are resolved.
4. In the **DonateMDBBean.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 msg) {
      handleMessage(msg);
   }
   ```
5. Save and close **DonateMDBBean.java.**
6. Work around: Publish the updated application (per the problem described in section "15.3.1 Work around: Disable automatic publishing" on page 370):
   - Switch to the **Servers** view, and observe that the status of **ExperienceJ2EE Server @ localhost** is **Republish**.
   - Select **ExperienceJ2EE Server @ localhost**, and right-click **Publish**.
   - Wait for the status of **ExperienceJ2EE Server @ localhost** to change to **Synchronized**.
7. Switch to the **Console** view, and verify that DonateEAR restarted correctly after completing these changes.

Behind the scenes:

- 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):

```
StringTokenizer tokens = new StringTokenizer(text, " ", false);
int employee_num =  Integer.parseInt(tokens.nextToken());
String fund_key = tokens.nextToken();
int donation_amount = Integer.parseInt(tokens.nextToken());
```

- Create an instance of the Donate session EJB (added by the **Call an EJB “create” method** wizard):

```
//* Insert call to EJB Create Method
Donate aDonate = createDonate();
```

- Invoke the donateVacation method, and print the result:

```
String resultMessage = aDonate.donateVacation(employee_num,
fund_name, donation_amount);
System.out.println("DonateMDB: call to donateVacation returned
" + resultMessage);
```

15.4 Create the messaging client (DonateMDBClient)

In this section, you create a J2EE application client that sends a JMS message to the Donate queue, thus allowing you to test the MDB.

1. Create the application client project
   - From the **Project Explorer**, select **Application Client Projects**, and right-click **New** → **Application Client Project**.
   - At the **Application Client Project** pop-up, perform the following actions:
     - **Name**: **DonateMDBClient**
     - **Target Runtime**: **WebSphere Application Server v6.1**
     - Select **Add project to an EAR**.
     - **EAR project**: **DonateEAR**
     - Click **Next**.
   - At the **Select Project Facets** pop-up, click **Next**.
   - At the **New Application Client Module** pop-up, ensure that **Create a default Main class** is selected, and click **Finish**.

2. From the **Project Explorer**, select **Application Client Projects** → **DonateMDBClient** → **appClientModule** → **(default package)** → **Main.java**, 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
   }
   ```

   – In the **Main.java** editor, clear out the "// TODO Auto-generated method stub" text to leave a blank line, and then position the cursor on the blank line.

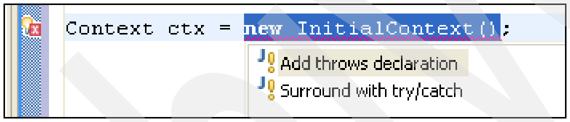
   – In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click on **FRAG11: DonateMDBClient**.

```
System.out.println("DonateMDBClient: Establishing initial context");
Context ctx = new InitialContext();

System.out.println("DonateMDBClient: Looking up ConnectionFactory");
ConnectionFactory cf = (ConnectionFactory)
ctx.lookup("jms/DonateCF");

System.out.println("DonateMDBClient: Lookup up Queue");
Destination outDest = (Destination) ctx.lookup("jms/DonateQ");

System.out.println("DonateMDBClient: Establishing connection");
Connection connection = cf.createConnection(“j2eeadmin”,”xxxxxxxx”);

System.out.println("DonateMDBClient: Establishing session");
Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
MessageProducer destSender = session.createProducer(outDest);

System.out.println("DonateMDBClient: Sending message");
TextMessage toutMessage = session.createTextMessage(args[0]);
destSender.send(toutMessage);

System.out.println("DonateMDBClient: Message sent!");
destSender.close();
session.close();
connection.close();
```

Figure 15-3 FRAG11: DonateMDBClient

- Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).
- At each **Organize Imports** pop-up, select the following imports and select **Next** (and at the last import select **Finish**):
  - javax.naming.Context
  - javax.jms.Connection
  - javax.jms.Destination
  - javax.jms.ConnectionFactory
  - javax.jms.MessageProducer
  - javax.jms.Session

  This resolves most but not all errors. We fix the remaining errors in the next step.
- At the remaining errors, left-click the quick fix icon [💡] in the left-hand margin, and accept the recommendation to **Add throws declaration**. You do this twice before all errors are resolved.



```
Context ctx = new InitialContext();
```

Behind the scenes:

These quick fix selections make the following code changes marked in bold:

```
import javax.jms.JMSException;
...
...
import javax.naming.NamingException;
...
public static void main(String[] args) throws
NamingException, JMSException
```

- Locate the following line:

```
Connection connection =
cf.createConnection("j2eeadmin","xxxxxxxx");
```

Change "xxxxxxxx" to match the password you configured for j2eeadmin in section 3.5, "Create the ExperienceJ2EE application server profile" on page 51. For example, if the password is PASSWORD you would change this line to the following:

```
Connection connection =
cf.createConnection(“j2eeadmin”,”PASSWORD”);
```

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 section 15.2.1, “Create the Service Integration Bus” on page 354).

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 did not supply a valid user ID and password, the createConnection operation fails with the following error:

```
WSCL0100E: Exception received:
java.lang.reflect.InvocationTargetException
...
...
Caused by: javax.jms.JMSException: CWSIA0241E: An
exception was received during the call to the method
JmsManagedConnectionFactoryImpl.createConnection:
com.ibm.websphere.sib.exception.SINotPossibleInCurrentCon
figurationException: CWSIT0008E: A successful connection
was made to the bootstrap server at
localhost:7276:BootstrapBasicMessaging but the server
returned an error condition: CWSIT0090E: A bootstrap
request was made to bus ExperienceJ2EE using channel
chain InboundBasicMessaging. Use of this chain is not
permitted by bus
```

- You could use any valid user ID for this connection because you configured the SIB to accept all authenticated users (in section 15.2.2, “Configure security” on page 358):
  - The j2eeadmin user ID defined in section “3.5 Create the ExperienceJ2EE application server profile” on page 51.
  - The DNARMSTRONG, DCBROWN, or DGADAMS user IDs defined in section 11.2, “Preliminary security setup” on page 261.

– Save and close.

Behind the scenes:

- Initialize the JNDI context:

```
Context ctx = new InitialContext();
```

- Locate the connection factory in the JNDI namespace:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup(
"jms/DonateCF");
```

- Locate the queue destination in the JNDI namespace:

```
Destination outDest = (Destination) ctx.lookup("jms/DonateQ");
```

- Create and start the connection:

```
Connection connection =
cf.createConnection(“j2eeadmin”,”xxxxxxxx”);
```

- 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();
```

Alternative:

For simplicity, the main method in the client application uses hard coded references to the JNDI names for the connection factory and the destination (Queue):

```
System.out.println("DonateMDBClient: Looking up
ConnectionFactory");
ConnectionFactory cf = (ConnectionFactory) ctx.lookup(
   "jms/DonateCF");
System.out.println("DonateMDBClient: Lookup up Queue");
Destination outDest = (Destination) ctx.lookup("jms/DonateQ");
```

This is acceptable in this simple example, but may be limiting in a production application that has to consider the possibility that other applications may have selected the same JNDI names. For example, it is possible that another application uses “jms/DonateQ” as the destination name.

Therefore, for production applications you have to be able to map to alternative JNDI names, which you can do by using an environment reference:

1. Update the client-application main method to call an environment reference (such as “java:comp/env/DonateQ”) instead of the JNDI name (such as “jms/DonateQ”):

   ```
   Destination outDest = (Destination)
   ctx.lookup("java:comp/env/DonateQ");
   ```

2. Update the client-deployment descriptor, which the Programmer can do when developing the application or the Application Server Administrator can do at deployment time to resolve the reference:

   – From the **Project Explorer** view, select **Application Client Projects** → **DonateMDBClient**, and double-click to open the **Client Deployment Descriptor** editor.

   – At the resulting **Client Deployment Descriptor** editor**,** switch to the **References** tab, and click **Add** (below references):

     - At the **Reference** pop-up, select **Message destination reference**, and click **Next**.
     - At the **Message destination reference** pop-up, click **New Destination…** (in the lower right).
     - At the **Message destination (create a message destination)** pop-up, enter a name of **DonateQ**, and click **Finish**.

- At the **Message destination reference** pop-up, select **DonateQ**, and click **Next.** The name is automatically populated with **DonateQ**.
- At the **Message destination reference details** pop-up, perform the following actions:
  - **Type**: **javax.jms.Queue**
  - **Usage**: **ConsumesProduces**
  - Click **Finish**.
- Back at the References screen, select **MessageDestRef DonateQ**, and in the lower-right, under **WebSphere Bindings** set the **JNDI name** to **jms/DonateQ**.
- Save and close the **Client Deployment Descriptor**.

15.5 Test the messaging client

The following steps are similar to the those used in section 10.5, “Test the application client” on page 251 and 11.6.2, “Update and test the J2EE application client test configuration” on page 295.

Work around information:

The status of the **ExperienceJ2EE Server @ localhost** test server shows as **Republish**. This indicates that the **DonateEAR** application changed, and that the test server is running an older version. Recall that you disabled automatic publishing in section 15.3.1, “Work around: Disable automatic publishing” on page 370.

However, this is acceptable because all your changes, since the last publish operation, were to a J2EE application client project. No changes exist in the server side (in EAR, EJB, or Web modules).

1. From the **Project Explorer**, select **Application Client Projects** → **DonateMDBClient**, and from the action bar select **Run** → **Run…**.
2. At the **Create, Manage, and run Configurations** pop-up, perform the following actions:
   - On the left, under **Configurations,** select **WebSphere v6.1 Application Client**, and select **New**:
   - On the resulting right pane, on the **Application** tab, perform the following actions:

    - **Name**: **DonateMDBClient**
    - **WebSphere Runtime**: **WebSphere Application Server V6.1**
    - **Enterprise application**: **DonateEAR**
    - **Application client module**: **DonateMDBClient**
    - Select **Enable application client to connect to a server.**
    - Select **Use specific serve**r, and then select **ExperienceJ2EE Server @localhost**.
  - On the right pane, switch to the **Arguments** tab, and perform the following actions:
    - Append “**1 DonationFund 1**” to the program argument string WITH THE QUOTES. Make sure that you leave a space between this and the existing arguments. The resulting program arguments string should be similar to the following:

      ```
      -CCverbose=true “1 DonationFund 1”
      ```
  - Windows: On the right, switch to the **Arguments** tab, and change this value under **VM arguments** from the following:

    ```
    -Dcom.ibm.CORBA.ConfigURL=file:C:/IBM/AppServer/profiles/AppSrv01/properties/sas.client.props
    ```

    Then change it to the following:

    ```
    -Dcom.ibm.CORBA.ConfigURL=file:C:/ExperienceJ2EE/ejbclient.props
    ```
  - Linux: On the right, switch to the **Arguments** tab and change this value under **VM arguments** from the following:

    ```
    -Dcom.ibm.CORBA.ConfigURL=file:/opt/IBM/AppServer/profiles/AppSrv01/properties/sas.client.props
    ```

    Then change it to the following:

    ```
    -Dcom.ibm.CORBA.ConfigURL=file:/home/<user>/ExperienceJ2EE/ejbclient.props
    ```

    where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.
  - Click **Apply** to save the changes, and then select **Run** to execute the application client.

3. AST V6.1 should automatically switch to the **Console** view in the lower-right pane, displaying the output of the application client.

4. At the **Console** view, verify that the Application Client completed successfully:

```
DonateMDBClient: Establishing initial context
DonateMDBClient: Looking up ConnectionFactory
DonateMDBClient: Lookup up Queue
DonateMDBClient: Establishing connection
DonateMDBClient: Establishing session
DonateMDBClient: Sending message
DonateMDBClient: Message sent!
```

5. Switch the **Console** view to the **ExperienceJ2EEServer @ localhost** output using the Console selection icon [🖥 ▾] from the action bar in the lower right, and verify that the message processed successfully (Note there may be other messages between these.):

```
SystemOut     O DonateMDB: Input Message = 1 DonationFund 1
SystemOut     O DonateMDB: call to donateVacation returned  Transfer
                             Successful
```

15.6 Explore!

This IBM Redbook implements messaging using the embedded JMS provider shipped with WAS V6.1. This implementation provides a robust implementation for WAS V6.1 to WAS V6.1 interaction, but it is not capable of supporting other environments, such as non-Java or non-JMS applications.

For those environments, IBM provides a product called WebSphere MQ, which provides both a JMS and non-JMS messaging infrastructure for a wide variety of clients and platforms.

Note that you can configure WAS V6.1 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
- WAS V6.1 InfoCenter: Interoperation using WebSphere MQ server:
  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.pmc.doc/tasks/tjfp0002_.html

16

Add publication of results

In this chapter you upgrade the message-driven bean to publish the results to a topic. You also create a J2EE application to subscribe to messages sent to this topic.

16.1 Learn!

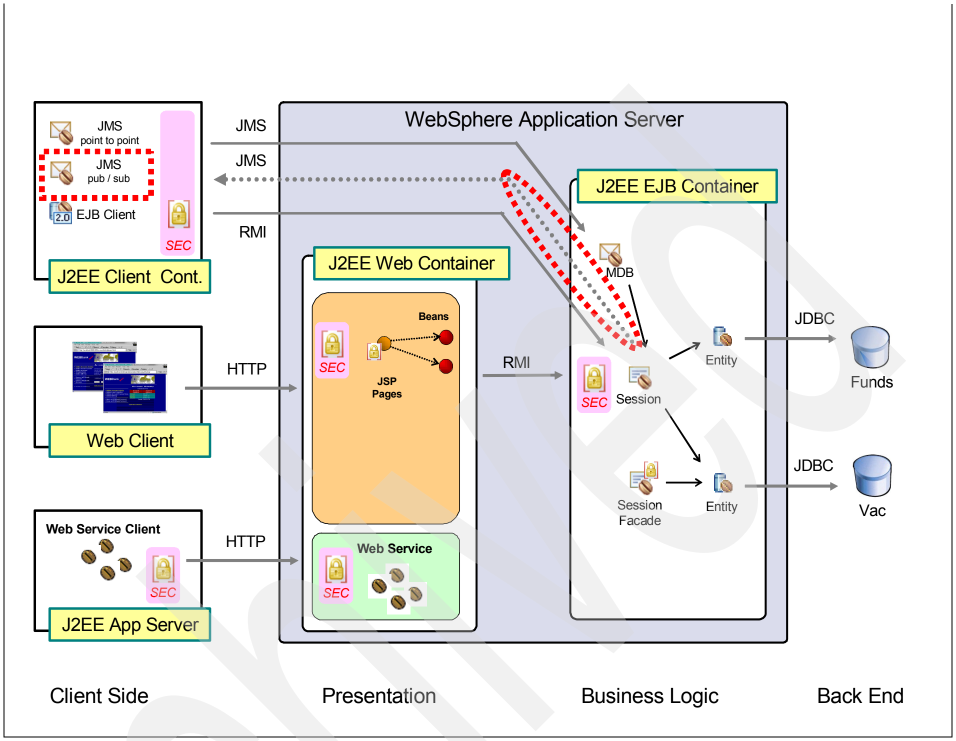


Figure 16-1 Donate application: Publication of results

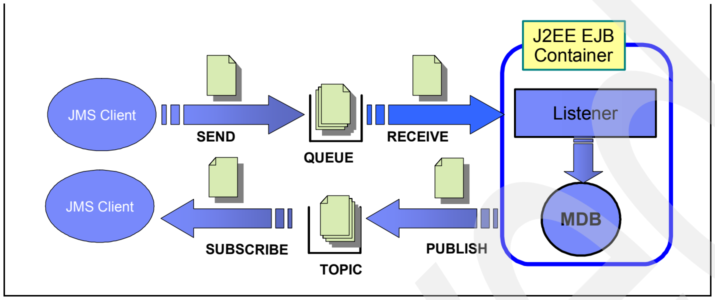
The point-to-point messaging used in the previous chapter assumes that the message is sent to a single known destination. What if you do not know the intended destination? What if multiple programs may require this message?

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 (like the 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.



There are no unique learn resources for this chapter.

16.2 Update the Donate session EJB

In this section you update the Donate session EJB to publish the results of a donateVacation invocation to a topic.

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Session Beans** → **Donate** → **DonateBean**(.java), and double-click to open.
2. Add the required global variables by performing the following tasks:
   - Add the following code, in bold, after "private javax.ejb.SessionContext mySessionCtx;".

```
public class DonateBean implements javax.ejb.SessionBean {

   private javax.ejb.SessionContext mySessionCtx;

   //* Added for pub sub logic
   Connection connection;
   Session session;
```

   - Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).
   - At each **Organize Imports** pop-up, select the following imports, and select **Next** (and at the last import select **Finish**):

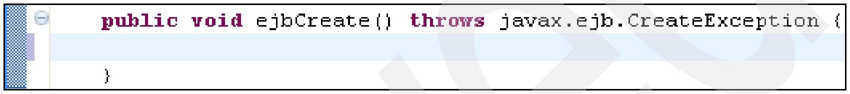
- javax.jms.Connection
- javax.jms.Session

Behind the scenes:

These two global variables are referenced in the ejbCreate method to initialize access to the connection and the (topic) session that are used to publish the results.

3. Update the ejbCreate method by performing the following actions:

   – Locate the **ejbCreate()** method, add a blank line in the method, and position the cursor at the blank line.



```
    public void ejbCreate() throws javax.ejb.CreateException {

    }
```

   – In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG12: Donate pubsub ejbCreate**.

```
//* Added for pub sub logic
try {
   InitialContext initContext = new InitialContext();
   ConnectionFactory cf = (ConnectionFactory)
initContext.lookup(“jms/DonateCF");
   connection = cf.createConnection();
   session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
   connection.start();
} catch (Exception e) {
   throw new EJBException("Donate: error on pub sub logic in
ejbCreate", e);
}
```

Figure 16-2 FRAG12: Donate pubsub ejbCreate

   – Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).

   – At the **Organize Imports** pop-up, select **javax.jms.ConnectionFactory**, and click **Finish**.

Behind the scenes:

- Initialize the JNDI context:

```
Context ctx = new InitialContext();
```

- Locate the Connection Factory in the JNDI namespace:

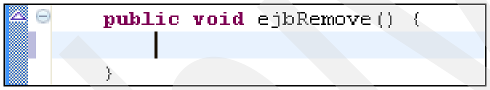
```
ConnectionFactory cf = (ConnectionFactory)
initContext.lookup("jms/DonateCF”);
```

- Initialize the connection and session variables:

```
connection = cf.createConnection();
session = connection.createSession(false,
 Session.AUTO_ACKNOWLEDGE);
connection.start()
```

4. Update the ejbRemove method by performing the following actions:

   – Locate the **ejbRemove()** method, add a blank line in the method, and position the cursor at the blank line.



```
    public void ejbRemove() {

    }
```

   – In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click on **FRAG13: Donate pubsub ejbRemove**.

```
//* added for pub sub logic
try {
   if (session != null)
      session.close();
   if (connection != null)
      connection.close();
} catch (JMSException e) {
   e.printStackTrace();
}
```

Figure 16-3 FRAG13: Donate pubsub ejbRemove

   – Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).

Behind the scenes:

When the ejb is de-instantiated, this code releases the resources associated with the session and connection variables.

5. Add the publishResults method by performing the following tasks:

   – Insert a blank line before the closing brace in the file, and position the cursor on this new line. Be sure to add the blank line before the closing brace for the overall file and not before the closing brace for the donateVacation method.

   – In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG14: Donate pubsub publishResults**.

```
//* added for pub sub
private void publishResults(int employee_num, String fund_name, int
donation_amount, String return_string) {
try {
   Destination emp1Topic =
session.createTopic(String.valueOf(employee_num));
   MessageProducer empPub = session.createProducer(emp1Topic);
   TextMessage pubMessage = session.createTextMessage("Request = " +
employee_num + " " + fund_name + " " + donation_amount + " , result
= " + return_string);
   empPub.send(pubMessage);
   System.out.println("Message sent to " + emp1Topic + " : " +
pubMessage.getText());
   empPub.close();
} catch (Exception e) {
   System.out.println("Error on Pub sub:" +
e.getLocalizedMessage());
}
}
```

Figure 16-4 FRAG14: Donate pubsub publishResults

   – Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).

   – At each **Organize Imports** pop-up, select the following imports, and select **Next** (and at the last import select **Finish**):

     - javax.jms.Destination
     - javax.jms.MessageProducer

Behind the scenes:

- Create a (topic) destination for the employee:

```
Destination emp1Topic =
   session.createTopic(String.valueOf(employee_num));
```

- Create a producer for the destination:

```
MessageProducer empPub = session.createProducer(emp1Topic);
```

- Create the message:

```
TextMessage pubMessage = session.createTextMessage(
     "Request = " + employee_num + " " + fund_name + " " + ""
    + donation_amount + " , result = " + return_string);
```

- Publish the message:

```
empPub.send(pubMessage);
```

- Free up resources:

```
empPub.close();
```

6. Update the donateVacation method by performing the following tasks:

   – To call the publishResults method, add the following code in bold to the **donateVacation** method just before **return return_string;**

```
public String donateVacation(int employee_num, String fund_name,
      int donation_amount) {
...
...
         }
      }
   }
   // * Added for pub sub logic
   publishResults(employee_num, fund_name, donation_amount,
return_string);

   return return_string;
}
```

7. Save and close **DonateBean.java**.

8. Work around: Publish the updated application (per the problem described in section ”15.3.1 Work around: Disable automatic publishing” on page 370) by performing the following actions:

   – Switch to the **Servers** view, and observe that the status of **ExperienceJ2EE Server @ localhost** should be **Republish**.

- Select **ExperienceJ2EE Server @ localhost**, and right-click **Publish**.
- Wait for the status of **ExperienceJ2EE Server @ localhost** to change to **Synchronized**.

9. Switch to the **Console** view, and verify that the **DonateEAR** application restarted correctly after completing these changes.

16.3 Create the messaging pub sub client (DonatePubSubClient)

These steps are similar to those used in section 15.4, "Create the messaging client (DonateMDBClient)" on page 377.

1. Create the application-client project by performing the following tasks.
   - From the **Project Explorer**, select **Application Client Projects**, and right-click **New** → **Application Client Project**.
   - At the **Application Client Project** pop-up, perform the following actions:
     - **Name**: **DonatePubSubClient**
     - **Target Runtime**: **WebSphere Application Server v6.1**
     - Select **Add project to an EAR**.
     - **EAR project**: **DonateEAR**
     - Click **Next**.
   - At the **Select Project Facets** pop-up, click **Next**.
   - At the **Application Client Module** pop-up, ensure that **Create a default Main class** is selected, and click **Finish**.
2. From the **Project Explorer**, select **Application Client Projects** → **DonatePubSubClient** → **appClientModule** → **(default package)** → **Main.java**, 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
}
```

   - In the **Main.java** editor, clear out the // TODO Auto-generated method stub text to leave a blank line, and position the cursor on the blank line.
   - In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG15: DonatePubSubClient**.

```
System.out.println("DonatePubSubClient: Establishing initial
context");
Context ctx = new InitialContext();

System.out.println("DonatePubSubClient: Looking up
ConnectionFactory");
ConnectionFactory cf = (ConnectionFactory)
ctx.lookup("jms/DonateCF");

System.out.println("DonatePubSubClient: Establishing connection");
Connection connection = cf.createConnection("j2eeadmin","xxxxxxxx");
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();
```

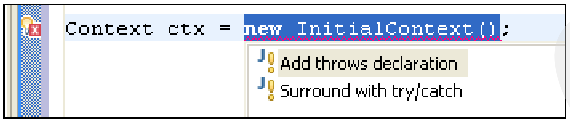
Figure 16-5 FRAG15: DonatePubSubClient

- Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).
- At each **Organize Imports** pop-up, select the following imports, and select **Next** (and at the last import select **Finish**):
  - javax.naming.Context
  - javax.jms.Destination

- javax.jms.Connection
- javax.jms.ConnectionFactory
- javax.jms.Session

This fixes most but not all errors. We fix the remaining errors in the step.

– At the remaining errors, left-click the quick fix icon [💡] in the left-hand margin, and accept the recommendation to Add throws declaration. You must do this twice before all errors are resolved.



```
Context ctx = new InitialContext();
```

– Locate the following line:

```
Connection connection =
cf.createConnection(“j2eeadmin”,”xxxxxxxx”);
```

Change “xxxxxxxx” to match the password you configured for j2eeadmin in section 3.5, “Create the ExperienceJ2EE application server profile” on page 51. For example, if the password is PASSWORD, change this line to the following:

```
Connection connection =
cf.createConnection(“j2eeadmin”,”PASSWORD”);
```

– Save and close.

Behind the scenes:

- Initialize the JNDI context:

```
Context ctx = new InitialContext();
```

- Locate the connection factory in the JNDI namespace:

```
ConnectionFactory cf =
   (ConnectionFactory) ctx.lookup("jms/DonateCF");
```

- Create/Start the connection:

```
Connection connection = cf.createConnection(“j2eeadmin”,
”xxxxxxxx”);
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();
```

16.4 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 section 16.5, “Explore!” on page 402 for information on both durable subscriptions and a related topic of message persistence.

16.4.1 Start DonatePubSubClient

The following steps are similar to the steps used in section15.5, “Test the messaging client” on page 383.

Work around information:

The status of the **ExperienceJ2EE Server @ localhost** test server shows as **Republish**. This indicates that the **DonateEAR** application changed, and that the test server is running an older version. Recall that you disabled automatic publishing in section 15.3.1, “Work around: Disable automatic publishing” on page 370.

However, this is acceptable because all your changes, since the last publish operation, were to a J2EE application-client project. No changes exist in the server side (EAR, EJB, or Web modules).

1. From the **Project Explorer**, select **Application Client Projects** → **DonatePubSubClient**, and from the action bar select **Run** → **Run…**
2. At the **Create, Manage, and run Configurations** pop-up, perform the following actions:
   - On the left, under **Configurations,** select **WebSphere v6.1 Application Client**, and select **New**:
   - On the resulting right pane of the **Application** tab, perform the following actions:
     - **Name**: **DonatePubSubClient**
     - **WebSphere Runtime**: **WebSphere Application Server V6.1**
     - **Enterprise application**: **DonateEAR**
     - **Application client module**: **DonatePubSubClient**
     - Select **Enable application client to connect to a server**.
     - Select **Use specific server**, and then select **ExperienceJ2EE Server @localhost**.
   - On the right pane, switch to the **Arguments** tab, and perform the following actions:
     - Append 1 to the program argument string, making sure that you leave a space between this and the existing arguments. The resulting program arguments string should be similar to the following:

       ```
       -CCverbose=true 1
       ```

– Windows : On the right, switch to the **Arguments** tab, and change this value under **VM arguments** from the following:

```
-Dcom.ibm.CORBA.ConfigURL=file:C:/IBM/AppServer/profiles/AppSr
v01/properties/sas.client.props
```

Change the above value to the following:

```
-Dcom.ibm.CORBA.ConfigURL=file:C:/ExperienceJ2EE/ejbclient.pro
ps
```

– Linux : On the right, switch to the **Arguments** tab, and change this value under **VM arguments** from the following:

```
-Dcom.ibm.CORBA.ConfigURL=file:/opt/IBM/AppServer/profiles/App
Srv01/properties/sas.client.props
```

Change the above value to the following to:

```
-Dcom.ibm.CORBA.ConfigURL=file:/home/<user>/ExperienceJ2EE/ejb
client.props
```

where <user> is the username you are currently running under. The combined value of /home/<user> should match the value of the $HOME environment variable.

– Click **Apply** (in the lower right of the pop-up) to save the changes (from both the Application and Arguments tab).

– Click **Run** to execute the Application Client.

3. AST V6.1 automatically switches to the **Console** view in the lower-right pane, displaying the output of the Application Client.

4. Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [🖥 ▾] from the action bar in the lower right. 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
```

16.4.2 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:

   – From the AST V6.1 action bar select **Run** → **Run…**

   – On the left, select **WebSphere v6.1 Application Client** → **DonateMDBClient**, and in the lower right click **Run**. This submits a message for employee 1 for DonationFund for 1 day.

   – Switch the **Console** view to the **DonateMDBClient** output using the Console selection icon [🖥▾] from the action bar in the lower right, and then verify that it sent a message.

```
DonateMDBClient: Establishing initial context
DonateMDBClient: Looking up ConnectionFactory
DonateMDBClient: Lookup up Queue
DonateMDBClient: Establishing connection
DonateMDBClient: Establishing session
DonateMDBClient: Sending message
DonateMDBClient: Message sent!
```

   – Switch the **Console** view to the **ExperienceJ2EE Server @ localhost** output using the Console selection icon [🖥▾] from the action bar in the lower right. Then verify that the MDB received the message and called donateVacation (and that donateVacation published the results).

```
SystemOut     O DonateMDB: Input Message = 1 DonationFund 1
SystemOut     O Message sent to topic://1 : Request = 1
DonationFund 1 , result = Transfer Successful
SystemOut     O DonateMDB: call to donateVacation returned
Transfer Successful
```

   – Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [🖥▾] from the action bar in the lower right, 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.

   – From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **enterEmployee.jsp**, and right-click **Run** → **Run on Server…**

- At the **enterEmployee.jsp** page, enter an **employee number** of **1**, and click **Lookup**.
- At the **employeeDetail.jsp** page, enter a **donation amount** of **1** (or any value), and click **Donate**.
- Switch the **Console** view to the **ExperienceJ2EE Server @ localhost** output using the Console selection icon [🖥 ▾] from the action bar in the lower right. Verify that the MDB received the message and called donateVacation (and that donateVacation published the results).

  ```
  SystemOut     O Message sent to topic://1 : Request = 1
  DonationFund 1 , result = Transfer Successful
  ```

- Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [🖥 ▾] from the action bar in the lower right, and then verify that it received the result message.

  ```
  returned message = Request = 1 DonationFund 1 , result = Transfer
  Successful
  ```

- Close the Web browser.

3. Submit a request for employee 2 through the Web interface by performing the following actions.
   - From the **Project Explorer**, select **Dynamic Web Projects** → **DonateWeb** → **WebContent** → **enterEmployee.jsp**, and right-click **Run** → **Run on Server…**
   - At the **enterEmployee.jsp** page, enter an **employee number** of **2**, and click **Lookup**.
   - At the **employeeDetail.jsp** page, enter a **donation amount** of **1** (or any value), and click **Donate**.
   - Switch the **Console** view to the **ExperienceJ2EE Server @ localhost** output using the Console selection icon [🖥 ▾] from the action bar in the lower right. Verify that the MDB received the message and called donateVacation (and that donateVacation published the results).

     ```
     SystemOut     O Message sent to topic://2 : Request = 2
     DonationFund 1 , result = Transfer Successful
     ```

   - Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [🖥 ▾] from the action bar in the lower right. Next, verify that it did NOT receive the result message because the **DonatePubSubClient** is subscribing to updates for employee 1, NOT for employee 2.
   - Close the Web browser.
4. Terminate the **DonatePubSubClient**.

- Switch to the **DonatePubSubClient** output in the **Console** view, and click the terminate icon [🟥].

16.5 Explore!

Section 15.6, “Explore!” on page 385 discussed how WAS V6.1 can utilize WebSphere MQ instead of the embedded JMS provider for base messaging transport. In a similar manner, WAS V6.1 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. Collectively these additional capabilities are called an Enterprise Service Bus (ESB). Refer to Chapter 18, “What next?” on page 415 for additional information on ESBs.

JMS messages are maintained asychronously 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. WAS V6.1 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:

- WAS V6.1 InfoCenter: Publishing and subscribing with a WebSphere MQ network:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.pmc.doc/concepts/cjc0005_.html

- WAS V6.1 InfoCenter: Using durable subscriptions:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.pmc.doc/tasks/tjn0012_.html

- WAS V6.1 InfoCenter: Learning about file stores:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.pmc.doc/tasks/tjm2000_.html

17

Implement security for messaging

In this chapter you update the MDB (DonateMDB) and the J2EE application client (DonateMDBClient) to operate with J2EE application security enabled.

17.1 Learn!

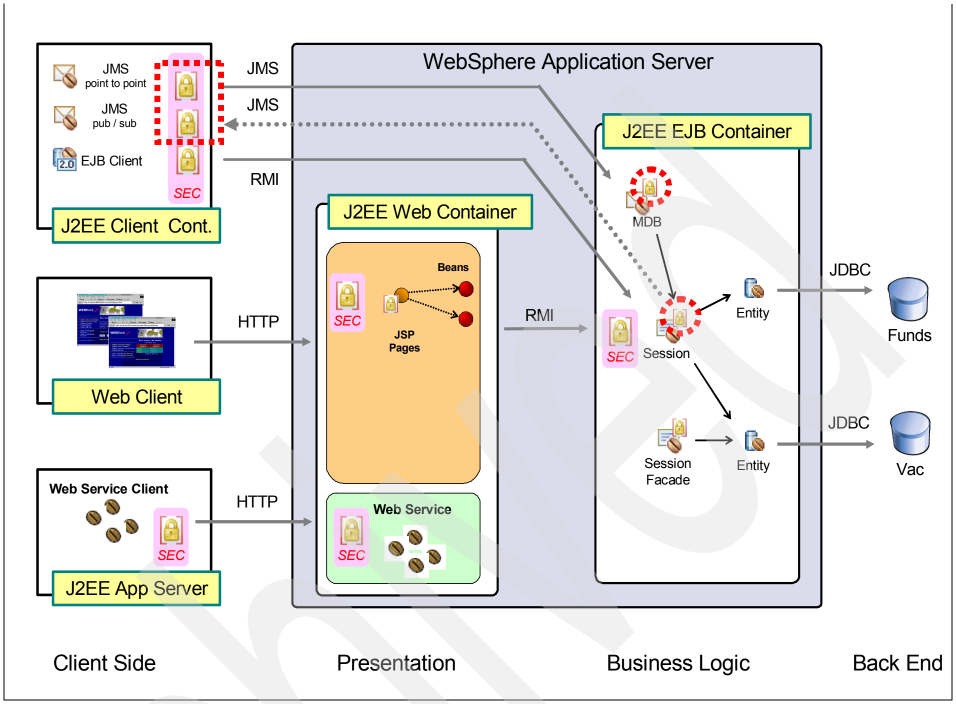


Figure 17-1 Donate application: Message security

Messaging has two levels of security. The first level controls interaction with the messaging provider to send and receive messages, which you already configured in the previous chapter when you performed the following actions:

- Creating the service integration bus (SIB) with security enabled in section ”15.2.1 Create the Service Integration Bus” on page 354.
- Adding a default role to the SIB that allows any authorized user to send and receive messages in section ”15.2.2 Configure security” on page 358.
- Configuring the activation specification (for the MDB) with a JAAS authentication alias in section ”15.2.5 Create the JMS activation specification” on page 365.
- Configuring the connection factory (for the server side pub sub code in the Donate session EJB donateVacation method) with a JAAS authentication alias in section ”15.2.6 Create the JMS connection factory for the application client” on page 367.

- Configuring the createConnection method in the two “messaging” J2EE application clients (DonateMDBClient, DonatePubSubClient) with user IDs and passwords in section ”15.4 Create the messaging client (DonateMDBClient)” on page 377 and in section ”16.3 Create the messaging pub sub client (DonatePubSubClient)” on page 394.

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 J2EE declarative security or programmatic security.

J2EE 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 the Run-As-Identity, which is defined through an entry in the EJB deployment descriptor, which points to a JAAS Authentication Alias (defined in the J2EE runtime environment) containing the user ID and password. The DonateMDB runs successfully without code changes if you added a Run-As Identity.

Both of these J2EE 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.

This scenario uses a user ID and password for simplicity, but you could also use a supported credential token format, such as an LTPA token.

The advantage of this approach (over the two J2EE provided alternatives) is that you maintain the integrity of the J2EE role-based security. The drawback is that users accessing your program must supply a user ID password in the JMS properties.

There are no unique learn resources for this chapter. Instead, refer to the security specific sections in the learn resources referenced in section 15.1, “Learn!” on page 350.

17.2 Re-enable J2EE application security

In this section you perform the complete set of tasks needed to switch a Test Server and Workspace combination from running with J2EE application security disabled to a combination running with J2EE application security enable.

1. Re-enable J2EE application security using the same steps described in section ”14.2 Re-enable J2EE application security” on page 334. Reference the following sections:
   - “14.2.1 Re-enable J2EE application security”
   - “14.2.2 Restart the ExperienceJ2EE Server @ localhost test server”

17.3 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 J2EE application security enabled.

1. From the AST V6.1 action bar, select **Run** → **Run…**
2. On the left, select **WebSphere v6.1 Application Client** → **DonateMDBClient**, and in the lower right click **Run**. This submits a message for employee 1 for DonationFund for 1 day.
3. Switch the **Console** view to the **DonateMDBClient** output using the Console selection icon [🖥▾] from the action bar in the lower right, and then verify that it sent a message.

```
DonateMDBClient: Establishing initial context
DonateMDBClient: Looking up ConnectionFactory
DonateMDBClient: Lookup up Queue
DonateMDBClient: Establishing connection
DonateMDBClient: Establishing session
DonateMDBClient: Sending message
DonateMDBClient: Message sent!
```

4. Switch the **Console** view to the **ExperienceJ2EE Server @ localhost** output using the Console selection icon [🖥▾] from the action bar in the lower right. Note that the Donate session EJB failed to process the message because the user context was UNAUTHENTICATED:

```
SystemOut     O DonateMDB: error on call to donateVacation:CORBA
NO_PERMISSION 0x0 No; nested exception is:
   org.omg.CORBA.NO_PERMISSION: java.rmi.AccessException:  ; nested
exception is:
   com.ibm.websphere.csi.CSIAccessException: SECJ0053E:
Authorization failed for /UNAUTHENTICATED while invoking
(Bean)ejb/sample2/DonateHome
donateVacation(int,java.lang.String,int):1 securityName:
/UNAUTHENTICATED;accessID: UNAUTHENTICATED is not granted any of the
required roles: DUSERS_ROLE DMANAGERS_ROLE vmcid: 0x0  minor code: 0
completed: No
```

17.4 Update DonateMDBClient and DonateMDB

In this section you update the DonateMDBClient to put the user ID and password in the JMS header. You also update the DonateMDB onMessage method to extract these values, and then execute the donateMethod as that user.

17.4.1 Update the DonateMDBClient code

1. From the **Project Explorer**, select **DonateMDBClient** → **appClientModule** → **(default package)** → **Main.java**, and double-click to open.
2. Before the **destSender.send(xxx)** call, add the following statements (in bold) to set properties in the JMS header that specify the user ID and password on whose behalf the MDB should execute the Donate session EJB (for example, in your case either DNARMSTRONG or DCBROWN).

   ```
   TextMessage toutMessage = session.createTextMessage(args[0]);

   //* Added for MDB Security
   toutMessage.setStringProperty("userid", "DNARMSTRONG");
   toutMessage.setStringProperty("password","xxxxxxxx");

   destSender.send(toutMessage);
   ```

   where “xxxxxxxx” is the password for DNARMSTRONG (or DCBROWN).
3. Save and close.

17.4.2 Update the DonateMDB code

1. From the **Project Explorer**, select **EJB Projects** → **DonateEJB2** → **DonateEJB2** → **Message-Driven Beans** → **DonateMDB** → **DonateMDBBean**(.java), and double-click to open.
2. Locate the **onMessage** method, comment out the **“handleMessage(msg);”** line by adding // at the beginning. Insert a blank line below, and position the cursor at this blank line.

```
public void onMessage(javax.jms.Message msg) {
   //handleMessage(msg);
}
```

3. In the **Snippets** view (lower right), expand the **Experience J2EE** category, and double-click **FRAG16: DonateMDB security**.

```
final Message msg2 = msg;
try {
   String tuserid = msg.getStringProperty("userid");
   String tpassword = msg.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 J2EE resource using the authenticated subject
   WSSubject.doAs(lc.getSubject(), new
java.security.PrivilegedAction() {
      public Object run() {
         handleMessage(msg2);
         return null;
      }
   });
} catch (Exception e) {
   System.out.println("DonateMDB: Exception: " +
e.getLocalizedMessage());
   e.printStackTrace();
}
```

Figure 17-2 FRAG16: DonateMDB security

   – Format the code (**Ctrl+Shift+F** on Windows or **Esc Ctrl-F** on Linux), and organize the imports (**Ctrl+Shift+O**).

   - At the **Organize Imports** pop-up, select **javax.jms.Message**, and click **Finish**. All errors should be resolved.

4. Save and close.
5. Work around: Publish the updated application (per the problem described in section ”15.3.1 Work around: Disable automatic publishing” on page 370):
   - Switch to the **Servers** view, and observe that the status of **ExperienceJ2EE Server @ localhost** should be **Republish**.
   - Select **ExperienceJ2EE Server @ localhost**, and right-click **Publish**.
   - Wait for the status of **ExperienceJ2EE Server @ localhost** to change to **Synchronized**.
6. Switch back to the **Console** view, and look for the message indicating that the DonateEAR application restarted successfully. This may not be the last message, so search backwards for WSVR0221I to locate the following:

```
WSVR0221I: Application started: DonateEAR
```

Behind the scenes:

- 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(msg2);
      return null;
   }
});
```

17.5 Test the updated messaging artifacts

In this section you test the updated messaging artifacts to verify that the DonateMDB onMessage method successfully processes the inbound message with J2EE application security enabled.

1. From the AST V6.1action bar select **Run** → **Run...**
2. On the left, select **WebSphere v6.1 Application Client** → **DonatePubSubClient**, and in the lower right click **Run**.
3. Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [🖥▾] from the action bar in the lower right, and then verify that it is waiting for a message.

```
DonatePubSubClient: Establishing initial context
DonatePubSubClient: Looking up ConnectionFactory
DonatePubSubClient: Establishing connection
DonatePubSubClient: Establishing session
DonatePubSubClient: Receiving messages for updates to employee
topic://1
```

4. On the left, select **WebSphere v6.1 Application Client** → **DonateMDBClient**, and in the lower right click **Run**. This submits a message for employee 1 for DonationFund for 1 day.
5. Switch the **Console** view to the **DonateMDBClient** output using the Console selection icon [🖥▾] from the action bar in the lower right, and then verify that it sent a message.

```
   DonateMDBClient: Establishing initial context
   DonateMDBClient: Looking up ConnectionFactory
   DonateMDBClient: Lookup up Queue
   DonateMDBClient: Establishing connection
   DonateMDBClient: Establishing session
   DonateMDBClient: Sending message
   DonateMDBClient: Message sent!
```

6. Switch the **Console** view to the **ExperienceJ2EE Server @ localhost** output using the Console selection icon [🖥▾] from the action bar in the lower right. Verify that DonateMDB received the message, set the user context to DNARMSTRONG (or DCBROWN), and successfully invoked the Donate session EJB:

```
SystemOut     O DonateMDB: Subject set to Subject:
   Principal: defaultWIMFileBasedRealm/DNARMSTRONG
   Public Credential:
         com.ibm.ws.security.auth.WSCredentialImpl@8e808e8
   Private Credential:
         com.ibm.ws.security.token.SingleSignonTokenImpl@2dac2dac
   Private Credential:
         com.ibm.ws.security.token.AuthenticationTokenImpl@73027302
   Private Credential:
         com.ibm.ws.security.token.AuthorizationTokenImpl@2d022d02
SystemOut     O DonateMDB: Input Message = 1 DonationFund 1
...
...
SystemOut     O Message sent to topic://1 : Request = 1 DonationFund
1 , result = Transfer Successful
[SystemOut     O DonateMDB: call to donateVacation returned Transfer
Successful
```

7. Switch the **Console** view to the **DonatePubSubClient** output using the Console selection icon [🖥 ▾] from the action bar in the lower right, and 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
```

17.6 Disable J2EE application security-Optional

Since this is the last scenario in this book, you can leave the J2EE application security enabled. However, if you want to experiment with other applications or J2EE capabilities, disable the J2EE application so you can work on these new artifacts first without J2EE application security.

1. Disable J2EE application security using the same steps described in section ”11.7 Disable J2EE application security” on page 296. Reference the following sections:
   – “11.7.1 Disable J2EE application security”.
   – “11.7.2 Restart the ExperienceJ2EE Server @ localhost test server”.

17.7 Explore!

There are no unique explore resources for this chapter.

18

What next?

Congratulations! You completed the Experience J2EE! scenario: You created and tested a comprehensive J2EE application addressing real-life issues such as supporting multiple back-end databases, security, Web services, and messaging (JMS).

My hope is that this book met or exceeded your expectations, and that by *Experiencing J2EE* you were able to move from a theoretical knowledge gained by reading introductory material to a practical knowledge gained by implementing a real-life application.

However, the knowledge level imparted by this book is a starting point. You need to continue to grow the breadth and depth of your expertise in J2EE.

The Explore resources at the end of each chapter are intended to assist you in expanding your knowledge of the current technologies discussed in this book, but they deal with current aspects of J2EE, while J2EE is continuing to evolve.

What other areas should you explore to ensure that you are still current with J2EE and J2EE development in a year or two years?

In addition, J2EE forms a core foundation for a modern computing infrastructure, but it does not address all the requirements that exist. What other technologies should you consider?

This chapter provides a brief discussion of some of the key technologies and products you should consider investigating (in the author's opinion). Table 18-1 organizes these topics against the J2EE tiered architecture, which we discussed in section "1.2.5 Container architecture" on page 15.

Table 18-1 Sections discussing each J2EE tiered architecture

Tier	Focus section
Overall	► "18.1 developerWorks, alphaWorks, and Google" ► "18.2 Rational Application Developer" ► "18.5 Java EE 5"
Client Tier	► "18.3 Asynchronous JavaScript and XML"
Presentation Tier	► "18.4 Portals" ► "18.5.1 Java EE 5: JavaServer Faces"
Business Logic Tier	► "18.5.2 Java EE 5: Enterprise JavaBeans V3.0"
Back End Tier	► "18.5.3 Java EE 5: Web services support" ► "18.6 Enterprise Service Buses" ► "18.7 Business Process Execution Language" ► "18.8 Service Oriented Architecture"

18.1 developerWorks, alphaWorks, and Google

developerWorks, alphaWorks, and Google are obviously not specific technical topics, but they are key sources of information that can help you explore both J2EE and related technologies:

- **developerWorks** is the IBM primary Web site for current products and technologies. **alphaWorks** is the IBM primary Web site for emerging technologies and products. developerWorks, in particular, has varied content including articles, forums, blogs, user groups, lists of technical events and web casts, and links to other IBM technical content Web sites, such as the product support pages and IBM Redbooks.
  - IBM developerWorks:

    `http://www.ibm.com/developerworks`
  - IBM alphaWorks:

    `http://www.ibm.com/alphaworks`
- **Google** provides outstanding search capabilities, both for a preliminary search that often links you directly to the appropriate content on developerWorks (for example “IBM tutorials on JSF?”), or as a search of last resort in case the standard Web sites listed above turn up no results.
  - Google home page:

    `http://www.google.com`
  - Google groups:

    `http://groups.google.com`

18.2 Rational Application Developer

AST V6.1 provides a basic integrated development for WAS V6.1 and does not contain all the capabilities that a developer might utilize for developing complex applications. Therefore, it is worthwhile to examine Rational Application Developer V7.0 and its capabilities (such as visual EJB editors, SDO/SCA support, JSF support, UML support, and code analysis).

AST V6.1 is based on the Eclipse Web Tools Platform (WTP) project, and RAtional Application Developer V7.0 is in turn based on AST V6.1. Therefore, it is also worthwhile to stay informed on the evolution of the WTP effort.

Explore resources are available at the following Web sites:

- Eclipse Web Tools Platform Project:

  `http://www.eclipse.org/webtools`

- *Rational Application Developer V6 Programming Guide*, SG24-6449:

  `http://www.redbooks.ibm.com/abstracts/sg246449.htm`

- IBM developerWorks: Rational Application Developer for WebSphere Software:

  `http://www.ibm.com/developerworks/rational/products/rad`

18.3 Asynchronous JavaScript and XML

Consider a typical auto parts Web site where you progressively define the make, year, and model of your car. Using standard Web techniques, you have two options to implement this page:

- Download all the data with the initial page, and have the user progressively select the options using JavaScript. With this solution the initial page download is quite large because it must download all the data for all makes, years, and models.
- Download each set of data and after each selection refresh the page to the server to get the next set of data. With this solution the user must wait for the browser refresh after each selection.

Neither option provides ideal performance:

- The first option forces the user to wait for a large initial page download.
- The second option forces the user to endure multiple page refreshes.

How can you implement this page in a manner that provides acceptable user performance?

Asynchronous JavaScript and XML (AJAX) builds on top of standard capabilities, such as the JavaScript XMLHttpRequest, to allow a browser to refresh part of a page, based either on a user interaction or through a background browser-side event. This is sometimes called asynchronous Web design or interactive Web design.

AJAX is not a specification and is not part of J2EE (neither V1.4 nor V1.5). However, it is an evolving open source technology that addresses a key shortcoming in the standard browser interaction model and is gaining widespread attention and usage. Whether it becomes a de-facto “standard” like Struts remains to be seen.

Explore resources are available at the following Web sites:

- Asynchronous JavaScript Technology and XML (AJAX) With Java 2 Platform, Enterprise Edition:

  `http://java.sun.com/developer/technicalArticles/J2EE/AJAX`

- IBM alphaWorks: AJAX Toolkit Framework:

  `http://www.alphaworks.ibm.com/tech/ajaxtk`

18.4 Portals

Consider an environment where a user accesses multiple Web applications. These Web applications were all developed separately by different organizations and potentially might display related information—where the user might want to take a value from one application and use it to invoke a query in another application.

How do you allow these applications to operate in a single-user interface that uses the same security and access control model? How do you create a “user desktop” within the browser, similar to the windowed environments for normal desktops (like Windows, Macs, and Linux)?

The answer is that you implement a portal, which is a server-side application that provides a framework for presenting multiple applications within separate panes, on a single client-side browser page. Portals effectively provide “on the glass” application aggregation.

A portal environment requires two primary elements:

- Portlets, which are the display components for an application. In the J2EE environment, these can be based on special JSPs that utilize portlet tag libraries or Java-based classes that extend the GenericPortlet class (consider this to be similar to a servlet). Most vendors support both of these via *JSR 168: Portlet Specification SR 168*.

  JSP based portlets can also utilize JSF, allowing you to have a single pane that displays a sequence of related Web pages.

  Portlet messaging allows one portlet to be invoked based on a value from another portlet. For example, consider the automotive parts example cited in the AJAX section. The year selection could be configured as one portlet, which when set can then trigger the make selection in another portlet, which when set can then trigger the model selection in a third portlet. Other portlets could then be triggered as well, supplying additional data such as an image or a parts list.

AST V6.1 supports developing portlets using the JSR 168 portlet definition (Java based, not JSP based). Rational Application Developer V6.0 contains a full-portlet-development environment (JSP, Java based, JSF).

- A portal server that provides the common runtime environment that provides for security, user interface management, common display structure, and so forth.

  The portal server allows the administrator to define a series of pages, and on each page they can group one or more portlets. Access to these pages and portlets is controlled through role-based profiles, which are similar in concept to the J2EE application role-based security. As a result, each user sees a customized set of pages and portlets, providing them with a unique “desktop” for their role.

  WebSphere Portal V6.0 is the IBM current release, and it runs as a J2EE application on top of WebSphere Application Server V6.0 (not V6.1).

WAS V6.1 provides an environment that allows you to run portlets outside of a Portal Server. Thus, you can create one artifact that runs both outside of a portal, similar to a standalone JSP or servlet, and inside of a portal as a full-function portlet.

Explore resources are available at the following Web sites:

- JSR 168: Portlet Specification:

  `http://www.jcp.org/en/jsr/detail?id=168`

- IBM developerWorks: WebSphere Portal Zone:

  `http://www.ibm.com/developerworks/websphere/zones/portal`

18.5 Java EE 5

This latest version of the J2EE specification was finalized in May, 2006, and at some point the current J2EE V1.4 development tools and runtime environments will be migrated to Java EE 5. Therefore, it is worthwhile to examine what is coming in this new release. However, consider several aspects.

First, it will take some time before most major vendors have a base development and runtime environment that supports Java EE 5, and even longer until their infrastructure extensions can run on those releases.

Consider that Java EE 5 was finalized in May, 2006. Also consider the following IBM release cycle related to J2EE V1.4, where WebSphere Process Server is the IBM implementation of WS-BPEL on Application Server V6.0:

11/2003	J2EE V1.4 final specification release
12/2004	WebSphere Application Server V6.0 general availability
09/2005	WebSphere Process Server V6.0 general availability

Second, many of the changes contained in this release are aimed at simplifying the overall J2EE programming model and do not change the broader capabilities that were exploited in this document. For example, EJB 3.0 does change the underlying implementation mechanisms for entity EJBs, but the external view and capabilities stay basically the same. Since you created the entity EJB using a tooling view, presumably the tooling view for this under EJB 3.0 should be very similar.

Third, there are several key new capabilities such as JSF and the evolving WS standards that DO significantly change the external capabilities that are available to the developers. However, you do not need to wait for Java EE 5-based products to utilize these. Reference section 18.5.1 “Java EE 5: JavaServer Faces” and 18.5.3 “Java EE 5: Web services support”.

Explore resources are available at the following Web site:

- Java EE at a glance:

  http://java.sun.com/javaee

18.5.1 Java EE 5: JavaServer Faces

JSF capability is available today in Rational Application Developer V6.0 and WAS V6.1, so although this book did not use JSF, it is used for Web development.

When Rational Application Developer V7.0 is released with support for WAS V6.1, Chapter 9, “Create the Web front end” on page 213 of this book could easily be updated to use JSF with minimal impact to the other chapters.

Explore resources for this topic were listed in section ”9.5 Explore!” on page 243.

18.5.2 Java EE 5: Enterprise JavaBeans V3.0

As previously discussed, the external functional view of an EJB is virtually identical in both J2EE V1.4 (EJB V2.1) and Java EE 5 (EJB V3.0). For example, both have entity, session, and message-driven EJBs. Thus, the functional usage, as described in this book, remains effectively the same with EJB V3.0.

The key change between EJB V2.1 and EJB V3.0 is the implementation:

- EJB V2.1, used in this book, implements the EJB definitions in a combination of Java files and deployment descriptors.
- EJB V3.0 moves most of the EJB definitions out of deployment descriptors and into annotation (meta-data) that is inside the actual Java classes. In addition, EJB V3.0 simplifies the programming elements used to access EJBs.

These changes alter the actual programming artifacts that are generated by the IDE wizards when creating EJBs, and also alter the actual code that is inserted by the IDE wizards when inserting snippets to access EJBs. However, the functional capability and usage of these EJBs remains the same.

IBM has released a test implementation of the EJB V3.0 specification called the IBM WebSphere Application Server Version 6.1 Feature Pack for EJB 3. This alpha release can be used with AST V6.1 and WebSphere Application Server V6.1.

Explore resources are available at the following Web sites:

- Enterprise JavaBeans Technologies:

  `http://java.sun.com/products/ejb`

- developerWorks: The EJB Advocate: Is EJB 2.x dead yet?

  `http://www.ibm.com/developerworks/websphere/techjournal/0602_ejba/0602_ejba.html`

- IBM WebSphere Application Server Version 6.1 Feature Pack for EJB 3

  `https://www14.software.ibm.com/iwm/web/cc/earlyprograms/websphere/was61ejb3/`

18.5.3 Java EE 5: Web services support

The evolving WS standards and the supporting Java EE specifications are available with WAS V6.1 prior to Java EE 5 through the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services.

These standards provide capability beyond the basic Web services implementation described in Chapter 13, “Create the Web service” on page 305 and Chapter 14, “Implement security for the Web service” on page 331, and thus should have minimal impact on these chapters.

Explore resources for this topic were listed in section ”13.6 Explore!” on page 329.

18.6 Enterprise Service Buses

Consider the example of the donateVacation element used in this document, which allows a user to donate vacation in the increments of days. What if a related application wishes to use this function but it allows the user to submit the vacation in increments of hours? What if there are multiple applications that use increments of hours, but one is based on Web services and one is based on messaging? How do you handle these “disconnects” without writing unique custom code for each situation? The answer is an enterprise service bus (ESB).

An ESB is an infrastructure that connects dissimilar application requests, performing various operations to resolve the differences:

- Message transformation, taking a message format from the service requester and changing it to match the message format required by the actual service. These transformations can include field name changes, conversion between different units (such as hours to days), format changes (from text to XML for example), and field additions or field deletions.
- Protocol conversion, converting from one service technology, such as JMS, to another service technology, such as Web services or EJBs.
- Endpoint routing, delivering the service request to different instances of the back end service based on user-defined criteria, such as the identity of the requester.
- Quality of service conversion, such as changing a username and password security token on the service request into a security assertion markup language (SAML) security token required by the actual service.

ESBs represent a merging of technologies that have been available for years supporting messaging (JMS) and Web service environments. These technologies collectively enable the mediation of interaction between application components that have a “disconnect”.

The key concept is that the “disconnects” are addressed within the ESB:

- Without the ESB, these “disconnects” are implemented uniquely in each application, leading to the problem of *interface creep*, where the interface code and path length becomes quite large because of all the permutations that must be supported. This also causes a maintenance issue, because a change in one application causes changes in many other applications.

- With the ESB, these changes are implemented once in the ESB, without changes to the service requester and the back end application. When an application changes, a single-associated change is made in the ESB, which then makes the appropriate alteration to the service request to connect to the back end application.

IBM provides several products that include ESB functionality:

- **WebSphere Enterprise Service Bus V6.0** provides an ESB capability implemented on top of WebSphere Application Server V6.0. This product is well suited for environments that already have a J2EE infrastructure or are heavily focused on Web services and messaging (JMS).
- **WebSphere Message Broker V6.0** provides a non-J2EE ESB implementation that provides a broad set of connectivity both to Web services and messaging (JMS) as well as legacy connections such as FTP, sockets, and base messaging (non-JMS).
- **WebSphere Datapower Integration Appliance XI50** provides a hardware based ESB capability.

The service integration bus (SIB) in WAS V6.1 can also be described as an ESB, although typically its function is some what limited in comparison to the above products.

IBM has multiple ESB products because ESB is an architecture: The concepts and capabilities described by the ESB architecture are applicable to a broad range of environments, and therefore more than one product implementation is appropriate. In fact, there are many scenarios where you may choose to incorporate two or more of the above mentioned ESB products. One common usage pattern includes the following two:

- The Datapower appliance in a DMZ zone providing secure *edge of network* ESB support for service requests from external zones, such as converting the user-identity token from an external format to an internal format.
- The WebSphere Enterprise Service Bus in the core J2EE infrastructure provides the manipulation of the actual message and service requests needed to map formats from several applications into the expected format required by the back end service.

Explore resources are available at the following Web addresses:

- WebSphere Enterprise Service Bus product page:

  `http://www.ibm.com/software/integration/wsesb`

- *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212:

  `http://www.redbooks.ibm.com/abstracts/sg247212.html`

- WebSphere Message Broker product page:

  http://www.ibm.com/software/integration/wbimessagebroker
- WebSphere DataPower® Integration Appliance XI50 product page:

  http://www.ibm.com/software/integration/datapower/xi50

18.7 Business Process Execution Language

Portals address the need to aggregate applications *on the glass*. What if you have the requirement to aggregate application elements before reaching the end user?

Consider the example where you are able to look up a GPS coordinate anywhere in the world. What if you wanted to determine the current US State Departments travel advisories for the country represented by that coordinate? You would need to perform the following:

- Potentially convert the GPS coordinate from one representation to another because there are several ways of representing the GPS coordinates.
- Invoke an application that takes the GPS coordinate as an input and returns a country.
- Invokes another application that takes the country and returns the US State Department travel advisories.

You could implement this integration *on the glass* using portlets and portlet messaging. But what if you wanted to make this an actual application that the user could invoke? Put in the GPS coordinates (in any valid GPS coordinates) and get as output the US State Department travel advisories? The answer is to implement a solution that implements the Web Services Business Process Execution Language (WS-BPEL) specification, which provides for the coordinated execution of service elements.

The IBM product solution that implements WS-BPEL is WebSphere Process Server V6.0, which runs on Application Server V6.0, not on V6.1. The associated development tool is WebSphere Integration Developer V6.0, which is built on Rational Application Developer V6.0.

WebSphere Integration Developer provides a graphic development environment that allows the user to incorporate application services—defined both in WSDLs for traditional Web services and in service component architecture (SCA) files, for direct Java invocation and messaging invocation—into an organized calling sequence called a process. These processes can then be externally invoked through Web services, messaging, or direct Java.

Processes can be short running or long running:

- A short running process is similar in concept to a session EJB: All processing is done synchronously, and the process and associated application service elements all complete within a single, short-lived thread of execution. This type of process supports normal transactional coordination (commit and rollback).
- A long running process may take seconds, days, weeks, months, or years to complete. The overall process manager records the state of the process after invoking each application service element, and potentially can suspend the process if it reaches an element that is waiting for an external signal, such as a reply message from messaging, JMS, or a Web service request that contains a specific identifier called a correlation ID that indicates for which instance of the process that message is intended.

  A long-running process does not support normal transactional coordination, and instead you must use compensation (which is described below).

Note that a process needs to contain more than the calling sequence for normal execution. It also needs to control what happens when something goes wrong:

- Application services can return an exception condition, called a fault. Processes can be configured with fault handlers, allowing the developer to define alternate sequences that execute when a fault is detected, whether on an individual application-service element, on a group of application-service elements contained in a section of the process, or on the overall process.
- If an overall process fails (either due to a fault or through normal process navigation to a process element that indicates an end failing state), the process can attempt to reverse the results of the previous steps through two mechanisms:
  - It can use standard-transactional rollback if the process is short running, and if the application-service elements support transactional rollback.
  - It can use a special mechanism called compensation, where the process manager invokes specific application elements to reverse the previously completed steps.

    To use compensation, each application-service element must define two different operations: One for the normal (forward) invocation and one for the compensation (reverse) invocation.

    For example, if the forward operation deletes a data record, it could return a key to the process manager, representing a record in a temporary database that contains the deleted information. If compensation needs to be invoked, the process manager could pass this key to the compensation operation, which would use this key to retrieve the record from the temporary database and recreate the record.

IBM ships its WS-BPEL implementation with a feature called the Human Task Manager. This feature, when invoked from a process, adds an item to a task list, and then suspends the process. A user, based on user-defined roles, can view a list of available tasks and then claim and complete the specific task. This causes the process to be reactivated, and to continue to the next application-service element in the process.

IBM believes that this is a key element in a process-oriented or workflow-oriented solution: Business processes are composed of both automated and manual (human task) steps. Any product that provides process automation must support the integration of both the automated and manual steps.

Explore references are available at the following Web address:

- OASIS Web Services Business Process Execution Language (WS-BPEL) Technical Committee:

  `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`

- IBM developerWorks: WebSphere Process Server and WebSphere Integration Developer:

  `http://www.ibm.com/developerworks/websphere/zones/businessintegration/wps/wps.html`

- *Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6*, SG24-7205:

  `http://www.redbooks.ibm.com/abstracts/sg247205.html`

18.8 Service Oriented Architecture

The preceding products and technologies provide a wide range of function:

- Application infrastructures, such as J2EE (and .NET).
- Application connectivity technologies like Web services, messaging (both JMS and non-JMS), FTP, and sockets.
- Portals that provide “on the glass” integration.
- Enterprise Service Buses (ESBs) that manage the “disconnects” between application-service elements.
- Process engines based on Business Process Execution Language (WS-BPEL) that provide for the coordinated execution of application service elements.

Previous chapters in this book also touched on basic Web services capabilities like UDDI registries, common data interchange formats like XML and service data objects (SDOs), and invocation frameworks like service component architecture (SCA).

How do you combine these technologies into a comprehensive architecture that addresses the needs of an enterprise computing environment, not only for the runtime environment but also for the full life cycle needs, starting from the business-level definitions and ending with the actual deployed technical implementations? The answer is to use a service oriented architecture (SOA).

It is important to view SOA as an evolutionary step in defining a distributed computing environment rather than a revolutionary step. SOA throws nothing away. Instead, SOA defines an organized architecture around the existing standards and technologies that relate to invoking application service elements or services.

SOA is to services as the Internet or intranet is to TCP/IP networks:

- Services and TCP/IP networks represent the discreet but unmanaged elements. Without these elements, you could not have an SOA environment, and you could not have an Internet or intranet.
- SOA and an Internet or intranet represent the managed view of these discreet elements to form an overall enterprise environment.

  For an Internet or intranet these environments do not occur automatically; instead, support organizations and standards are put into place to coordinate and manage the changes that occur over time.

  The same is true for an SOA implementation: Organizations and internal standards are critical to ensure that the technologies are applied and maintained in a manner that provides a robust and reliable infrastructure.

Explore references are available at the following Web address:

- IBM developerWorks: IBM SOA Foundation: An architectural introduction and overview:

  http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper
- IBM developerWorks: SOA and Web services:

  http://www.ibm.com/developerworks/webservices
- *Enabling SOA Using WebSphere Messaging*, SG24-7163:

  http://www.redbooks.ibm.com/abstracts/sg247163.html
- *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240:

  http://www.redbooks.ibm.com/abstracts/sg247240.html

Part 5

Appendixes

A

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 Redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG247297`

Alternatively, you can visit the IBM Redbooks Web site at:

ibm.com/redbooks

After you access the Web site, select **Additional materials** and open the directory that corresponds with the redbook form number, SG247297.

Using the Web material

The additional Web material that accompanies this publication includes the following files:

File name	*Description*
ExperienceJ2EE.zip	ZIP file with 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

Please refer to section 2.2, “Hardware requirements” on page 24 and section 2.4, “Supported operating systems” on page 26 for further information.

How to use the Web material

Download the sample code and extract the ZIP file into a suitable directory. Refer to section 3.2, “Extract the ExperienceJ2EE.zip samples file” on page 44 for further instructions.

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 on ordering these publications, see “How to get IBM Redbooks” on page 435. Note that some of the documents referenced here may be available in softcopy only.

- *Rational Application Developer V6 Programming Guide*, SG24-6449
- *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257
- *WebSphere Application Server V6.1: Planning and Design*, SG24-7305
- *WebSphere Application Server V6.1 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/rationa

- WebSphere Information Center

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/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 and Web Tools Platform

  http://www.eclipse.org
- Web Tools Platform

  http://www.eclipse.org/webtools
- Eclipse plugin central

  http://www.eclipseplugincentral.com
- Sun Java

  http://java.sun.com
- Java Community Process

  http://www.jcp.org
- Apache Derby database

  http://db.apache.org/derby
- OASIS

  http://www.oasis-open.org
- mozilla.org

  http://www.mozilla.org
- Jython

  http://www.jython.org
- Apache Tuscany

  http://incubator.apache.org/tuscany
- Wikipedia

  http://en.wikipedia.org
- Google

  http://www.google.com
- Web Services Interoperability Organization

  http://www.ws-i.org

How to get IBM Redbooks

You can search for, view, or download IBM Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy redbooks or CD-ROMs, at the following Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Abbreviations and acronyms

AJAX	Asynchronous JavaScript Technology and XML
API	application programming interface
AST	Application Server Toolkit
BMP	bean managed persistence
CMP	container managed persistence
EGL	Enterprise Generation Language
EJB	Enterprise JavaBeans
ESB	Enterprise Service Bus
FAQ	frequently asked questions
FTP	File Transfer Protocol
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
ITSO	International Technical Support Organization
J2EE	Java 2, Enterprise Edition
J2SE	Java 2, Standard Edition
JAAS	Java Authentication and Authorization Service
JACC	Java Authorization Contract for Containers
JAX-RPC	Java API for XML-based RPC
JAX-WS	Java API for XML Web Services
JAXB	Java Architecture for XML Binding
JCA	Java Connector Architecture
JCP	Java Community Process
JDBC	Java Database Connectivity
JDK™	Java Developer’s Kit
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSF	JavaServer Faces
JSP	JavaServer Pages
JSR	Java Specification Request
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LTPA	Lightweight Third Party Authentication
MDB	message-driven bean
MQ	message queue
MTOM	Message Transmission Optimization Mechanism
MVC	model view controller
OASIS	Organization for the Advancement of Structure
RDB	relational database
RHEL	Red Hat Enterprise Linux
RMI	Remote Method Invocation
RPC	remote procedure call
SAAJ	SOAP with Attachments API for Java
SAML	Security Assertion Markup Language
SCA	Service Component Architecture
SDO	Service Data Objects
SIB	service integration bus
SIP	Session Initiation Protocol

SOA	service oriented architecture
SOAP	Simple Object Access Protocol (also known as Service Oriented Architecture Protocol)
SQL	structured query language
SSL	Secure Sockets Layer
SWAM	Simple WebSphere Authentication Mechanism
SWT	Standard Widget Toolkit
TCP/IP	Transmission Control Protocol/Internet Protocol
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
URI	uniform resource identifier
URL	uniform resource locator
UTC	Universal Test Client
VM	virtual machine
WS-BPEL	Web Services Business Process Execution Language
WS-I	Web Services Interoperability
WS-RM	Web Services Reliable Messaging
WSDL	Web Services Description Language
WSIL	Web Services Inspection Language
WTP	Web Tools Platform
WYSIWYG	what you see is what you get
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language

Index

IBM

Redbooks

Experience J2EE! Using WebSphere Application Server V6.1

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM®

Experience J2EE! Using WebSphere Application Server V6.1

Redbooks

Including Web services and messaging

Using the Application Server Toolkit

Using the Derby database

This IBM Redbook is a hands-on guide to developing a comprehensive J2EE application, including core functions, security, Web services, and messaging. In the context of J2EE, messaging is also known as Java Message Service (JMS).

Novice users are thus able to experience J2EE, 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 IBM Redbook also helps in those stages:

- Before experiencing J2EE, you *learn* about the base specifications and intellectual knowledge of J2EE through brief descriptions of the theory and through links to other information sources.
- After experiencing J2EE you will *explore* advanced J2EE through previews of advanced topics and links to other information sources.

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

SG24-7297-00 ISBN 0738494518