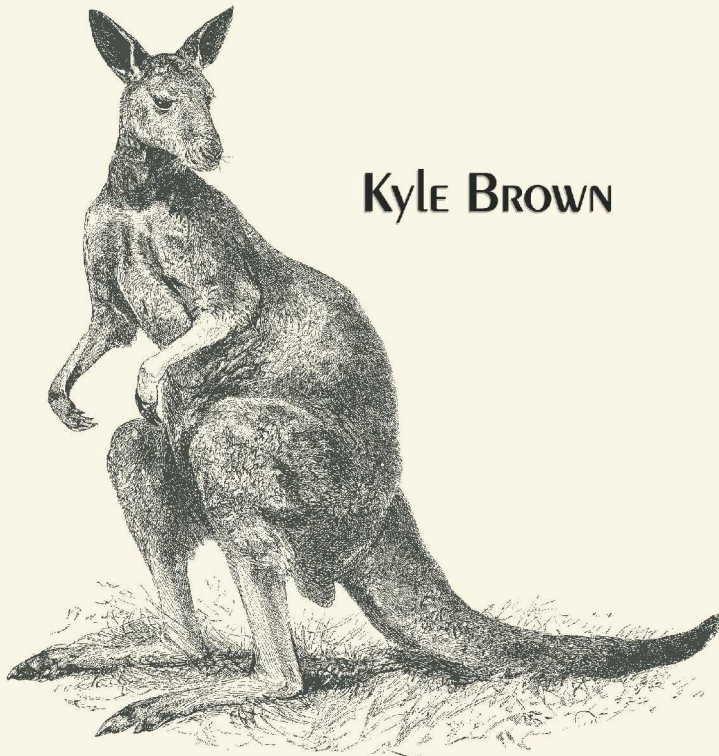


WebSphere™ 4.0 AEs

Workbook for ENTERPRISE JAVA BEANS, 3rd Edition



Kyle BROWN

COVERS
EJB 1.1

Windows
2000

Windows
NT

TITAN™
Books

Order the bound version of this book from <http://www.titan-books.com>

About the Series

Each of the books in this series is a server-specific companion to the third edition of Richard Monson-Haefel's best-selling and award-winning *Enterprise JavaBeans* (O'Reilly 2001), available at <http://www.titan-books.com/> and at all major retail outlets. It guides the reader step by step through the exercises called out in that work, explains how to build and deploy working solutions in a particular application server, and provides useful hints, tips, and warnings.

These workbooks are published by Titan Books in the context of a friendly agreement with O'Reilly and Associates, the publishers of *Enterprise JavaBeans*, to provide serious developers with the best possible foundation for success in EJB development on their chosen platforms.

Series Titles Available

WebLogic™ Server 6.1 Workbook for *Enterprise JavaBeans™ 3rd Edition*

WebSphere™ 4.0 AEs Workbook for *Enterprise JavaBeans™ 3rd Edition*

J2EE™ 1.3 SDK Workbook for *Enterprise JavaBeans™ 3rd Edition*

WebSphere™ 4.0 AEs Workbook
for
Enterprise JavaBeans™
3rd Edition

Kyle Brown



Order the bound version of this book from <http://www.titan-books.com>

WebSphere 4.0 AEs Workbook for *Enterprise JavaBeans, 3rd Edition*, by Kyle Brown

Published by Titan Books, Minneapolis, Minnesota.

Series Editor: Brian Christeson

Companion volume to *Enterprise JavaBeans, 3rd Edition*, by Richard Monson-Haefel, published by O'Reilly & Associates, Inc., 2001, available at <http://www.titan-books.com/>.

Copyright © 2001 Titan Books, Inc. All rights reserved.

Printed in the United States of America by Fidler Doubleday, Inc.

Printing History:

October 2001 First Edition

IBM, VisualAge, and WebSphere are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Other company, product, and service names may be trademarks or service marks of others.

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

ISBN 1-931822-47-6

Order the bound version of this book from <http://www.titan-books.com>

For Ann and Nathaniel

Table of Contents

Table of Figures.....	xi
Preface.....	xv
Contents of This Book.....	xv
On-Line Resources.....	xvi
Conventions Used in This Book.....	xvii
Acknowledgements.....	xvii
Server Installation and Configuration.....	1
Downloading and Installing the Software.....	1
Downloading IBM WebSphere Application Server AEs	1
Installing WebSphere AEs on Windows 2000 or Windows NT	2
Downloading and Installing IBM DB2.....	6
Before You Begin	7
Understanding the exercises in this workbook.....	7
Setting up the sample database.....	8
Setting DB2 to Use JDBC 2.0	11
An Introduction to J2EE and IBM WebSphere AEs.....	11
IBM WebSphere AEs Elements	14
Exercises for Chapter 4.....	25
Exercise 4.1: A Simple Entity Bean	26
Step 1: Compile the Java source code.....	26
Step 2: Create an EJB .jar file for the Cabin bean.....	27
Step 3: Create application client .jar files.....	27
Step 4: Use the AAT to assemble an enterprise archive (.ear) file.....	42
Step 5: Use the AAT to generate deployment code for the .ear file	48
Step 6: Use the DB2 Command Line Processor to create the Cabin bean table.....	52
Step 7: Use the Administrative Console to create a DB2 data source.....	54
Step 8: Use the Administrative Console to deploy the EJB .jar file.....	57
Step 9: Test your clients with the launchClient tool	64

Exercise 4.2: A Simple Session Bean	68
Step 1: Compile the TravelAgent bean classes and build an EJB .jar file	68
Step 2: Create an application client .jar file for the Client_3 test program.....	87
Step 3: Add the application client and the EJB to the .ear file	88
Step 4: Deploy the EJB .jar file to the server.....	90
Step 5: Run the Client_3 test program.....	91
Exercises for Chapter 5.....	93
Exercise 5.1: The Remote Component Interfaces	94
Step 1: Compile the classes	94
Step 2: Create three new application client .jar files.....	94
Step 3: Create the Client_51.ear file	95
Step 4: Test the application clients.....	95
Exercise 5.2: The EJBObject, Handle, and Primary Key.....	99
Step 1: Compile the classes	99
Step 2: Create three new application client .jar files.....	99
Step 3: Create the Client_53.ear file.....	100
Step 4: Test the application clients.....	100
Exercise for Chapter 9	103
Exercise 9.1: A CMP 1.1 Entity Bean	104
Step 1: Learn about WebSphere custom finder methods.....	104
Step 2: Compile the Ship classes	109
Step 3: Build the Ship EJB .jar file	109
Step 4: Build the application client .jar files	109
Step 5: Build the Ship .ear file	114
Step 6: Create the ShipEJB table.....	114
Step 7: Deploy the EJB .jar file for the Ship bean	114
Step 8: Run the Ship test clients.....	115
Exercise for Chapter 10.....	117
Exercise 10.1: A BMP Entity Bean	118
Step 1: Compile the Source Files	118
Step 2: Assemble the EJB .jar file.....	118
Step 3: Create the application client .jar files.....	118
Step 4: Assemble the .ear file and generate the deployed .ear file.....	119
Step 5: Create the Ship table.....	119

Step 6: Deploy the EJB .jar file into WebSphere AEs	119
Step 7: Test the new EJB with the test clients.....	120

***Exercises for Chapter 12* 121**

Exercise 12.1: A Stateless Session Bean 122

Step 1: Undeploy the existing Cabin and TravelAgent beans	125
Step 2: Compile the Source Files	126
Step 3: Assemble the EJB .jar file.....	127
Step 4: Create an application client .jar file	127
Step 5: Assemble the .ear file and generate the deployed .ear file.....	130
Step 6: Create the database tables.....	130
Step 7: Deploy the .ear file into WebSphere AEs	132
Step 8: Test the new EJBs with the test client	135

Exercise 12.2: A Stateful Session Bean.....137

Step 1: Compile the Source Files	137
Step 2: Assemble the EJB .jar file.....	137
Step 3: Create an application client .jar file	137
Step 4: Create the .ear files	140
Step 5: Use SEAppInstall to deploy the new TravelAgent .ear file	140
Step 6: Launch the application client.....	147

Table of Figures

Figure 1: Installation options	2
Figure 2: Security options	3
Figure 3: Product directories.....	4
Figure 4: Installation options selected.....	5
Figure 5: Restarting Windows.....	6
Figure 6: DB2 Universal Database First Steps.....	9
Figure 7: IBM DB2 CLP	10
Figure 8: Containers and components.....	12
Figure 9: Enterprise archive deployment structure.....	13
Figure 10: First Steps menu	15
Figure 11: Console with startup messages.....	16
Figure 12: Administrative Console login.....	17
Figure 13: Administrative Console – first startup	18
Figure 14: DB2 JDBC driver configuration.....	19
Figure 15: JDBC driver installation updated	20
Figure 16: Structure of WebSphere AEs.....	21
Figure 17: Saving your configuration	22
Figure 18: Console with application server stopped.....	23
Figure 19: Application Assembly Tool – start screen	28
Figure 20: Specifying module properties.....	29
Figure 21: Adding files.....	30
Figure 22: Selecting files to add	31
Figure 23: Selecting the root directory	32
Figure 24: Adding files – showing the dev directory	33
Figure 25: Adding Client_1	34
Figure 26: Adding files – showing Client_1.....	35

Figure 27: Specifying additional module properties.....	36
Figure 28: Selecting the main class.....	37
Figure 29: Choosing application icons.....	38
Figure 30: Adding EJB references	39
Figure 31: Adding environment entries	40
Figure 32: AAT showing completed client.....	41
Figure 33: Specifying application properties	42
Figure 34: Adding supplementary files.....	43
Figure 35: Adding EJB modules.....	44
Figure 36: Confirming values.....	45
Figure 37: Adding EJB modules – Cabin EJB .jar selected.....	45
Figure 38: Adding application clients.....	46
Figure 39: Adding security roles	47
Figure 40: AAT with EJB and client files.....	48
Figure 41: Generating code for deployment – start.....	49
Figure 42: Code generation successful.....	52
Figure 43: Data sources.....	54
Figure 44: Adding the sample data source	55
Figure 45: Data sources – to be saved.....	56
Figure 46: Saving your configuration file	57
Figure 47: Application Installation Wizard	58
Figure 48: Mapping roles to users and groups	59
Figure 49: Setting the JNDI name	60
Figure 50: Database settings	61
Figure 51: Confirming settings	62
Figure 52: Installed application list with Cabin application	63
Figure 53: Specifying EJB module properties	69
Figure 54: Adding files for the TravelAgent Bean.....	70
Figure 55: Adding enterprise beans	71
Figure 56: Choosing the bean type.....	72

Figure 57: Specifying EJB properties	72
Figure 58: Specifying EJB type properties.....	73
Figure 59: Adding EJB references – empty	74
Figure 60: Adding an EJB reference.....	75
Figure 61: Adding enterprise beans – TravelAgent added	76
Figure 62: Adding new security roles.....	77
Figure 63: Creating a security role	78
Figure 64: New security role added	79
Figure 65: Adding method permissions.....	80
Figure 66: Adding a method permission	81
Figure 67: Adding methods	82
Figure 68: Selecting a security role	83
Figure 69: Adding method permissions – completed	84
Figure 70: Adding container transactions	85
Figure 71: Adding a container transaction.....	86
Figure 72: Adding container transactions – completed	87
Figure 73: Importing the EJB module	88
Figure 74: Confirming the TravelAgent deployment descriptor	89
Figure 75: All clients and EJB modules added	90
Figure 76: Mapping ejb/CabinHomeRemote to a JNDI name.....	91
Figure 77: Setting resource references	120
Figure 78: Setting module visibility	123
Figure 79: Selecting applications to be uninstalled	125
Figure 80: Confirming applications to be uninstalled.....	126
Figure 81: Deploying multiple EJBs from an .ear file.....	133
Figure 82: Setting the redeployment option.....	134

Preface

This workbook is designed to be a companion for O'Reilly's *Enterprise Java Beans, 3rd Edition*, by Richard Monson-Haefel.

The goal of this WebSphere-specific workbook, like the other vendor-specific workbooks produced by Titan Books, is to provide practical, step-by-step instructions for installing and configuring IBM WebSphere Application Server, Advanced Edition Single-server (AEs) and deploying the example programs from the *Enterprise JavaBeans* book using this product.

This workbook also discusses key WebSphere-specific requirements, best practices, and the use of WebSphere-specific tools such as the Application Assembly Tool and the Administrative Console. You will learn how these tools interact, and how to test your example programs using WebSphere's J2EE Client Container, which is accessed through the launchClient application.

This book is based on the initial release of IBM WebSphere Application Server, Single-Server Edition (AEs), version 4.0 and includes all of the EJB 1.1 examples from the *Enterprise JavaBeans* book. All examples and techniques demonstrated in this book work properly with the 4.0 release of IBM WebSphere Application Server AEs, but not with previous versions of the product.

Contents of This Book

This workbook is divided into two major sections:

- ♦ **Server Installation and Configuration** – This section will walk you through downloading and installing WebSphere AEs, version 4.0, and also through downloading and installing IBM DB2 Personal Developer's Edition 7.2. It will also introduce some of the terminology of J2EE and WebSphere, and will help you properly configure both DB2 and your WebSphere Application Server for use with the exercises described in this book.
- ♦ **Exercises** – This section contains step-by-step instructions for downloading, building, configuring, and deploying the example programs for each of the EJB 1.1 exercises called out in *Enterprise JavaBeans, 3rd Edition* (which, for brevity, this workbook refers to as "the EJB book"). The text also examines many of the source files, descriptors, and other components used in each exercise to point out key WebSphere- and DB2-specific issues and techniques.

Because WebSphere AEs 4.0 is an EJB 1.1-compliant product, the EJB 2.0 exercises are not included in this workbook.

The workbook text for each exercise varies depending on the amount of new information presented in the exercise, and on the tasks required for setting up the example programs, but generally each exercise describes the following activities:

- ◆ Compiling and building the example code
- ◆ Assembling J2EE *client application jar files* and *enterprise archive files* with the Application Assembly Tool
- ◆ Creating any necessary database tables
- ◆ Deploying the EJB components to the application server
- ◆ Running the example programs and evaluating the results

The exercises have been designed to be built and run in order. There are dependencies both in the creation of executable files in one exercise that are needed in later exercises, and in the creation of data in the database used in multiple exercises. The instructions for the examples provide information as to when you should clean up the results of one exercise before beginning another.

On-Line Resources

This workbook is designed for use with the EJB book and with downloadable example code, both available from our web site:

<http://www.titan-books.com/>

We will post errata here, and any updates when required to support specification or product changes. This site also contains links to many popular EJB-related sites on the internet.

For help and information on using IBM WebSphere AEs, refer to the IBM Web site for WebSphere:

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

Another useful source for information is the Internet Newsgroup for the IBM WebSphere Application server:

<http://groups.google.com/groups?hl=en&safe=off&group=ibm.software.websphere.application-server>

Finally, an excellent resource for articles and best practices on using the IBM WebSphere Application server is the IBM WebSphere Developers's Domain (wsdd) web site:

<http://www.ibm.com/websphere/developer>

I hope you find this book useful in your study of Enterprise JavaBeans and the IBM WebSphere Application Server, AEs. Comments, suggestions, and error reports on the text of this workbook or the downloaded example files are welcome and appreciated. Please e-mail them to:

websphere-workbook@yahooogroups.com

Conventions Used in This Book

Italics are used for:

- ◆ Filenames and pathnames
- ◆ Names of hosts, domains, and applications
- ◆ URLs and email addresses
- ◆ New terms where they are defined

Boldface is used for:

- ◆ Emphasis
- ◆ Buttons, menu items, window and menu names, and other UI items you are asked to interact with

`Constant width` is used for:

- ◆ Code examples and fragments
- ◆ Sample program output
- ◆ Class, variable, and method names, and Java keywords used within the text
- ◆ SQL commands, table names, and column names
- ◆ XML elements and tags
- ◆ Commands you are to type at a prompt

`Constant width bold` is used for emphasis in some code examples.

`Constant width italic` is used to indicate text that is replaceable. For example, in *BeanNamePK*, you would replace *BeanName* with a specific bean name.

An Enterprise JavaBean consists of many parts; it's not a single object, but a collection of objects and interfaces. To refer to an Enterprise JavaBean as a whole, we use the name of its business name in Roman type followed by "bean" or the acronym "EJB." For example, we will refer to the Customer EJB when we want to talk about the enterprise bean in general. If we put the name in a constant width font, we are referring explicitly to the bean's class name, and usually to its remote interface. Thus `CustomerRemote` is the remote interface that defines the business methods of the Customer bean.

Acknowledgements

I would like to take time to say thank you to everyone who helped with the writing of this workbook, directly or indirectly. In particular:

Thanks to our series editor, Brian Christeson, for his hard work and perseverance.

Thanks to Richard Monson-Haefel for finally giving me the opportunity to collaborate with him on a project. It's been a fun experience.

Thanks to the terrific folks that I work with in IBM Software Services for WebSphere, including my managers, Lou Cipriano and Rich Hagopian, for supporting me in this effort, and the incredible team of reviewers from IBM that poured hours of their own time into reviewing this book. Those reviewers include Kirk Davis, Peter Xu, Adrian Spender, David Schlosser, and Ajit Kumar. Your efforts are greatly appreciated.

Finally, thanks to my darling wife Ann, who has put up with my many long nights at the computer while I developed this workbook, and my son Nathaniel, who has often missed his Daddy during its writing. I love you both.

Server Installation and Configuration

Downloading and Installing the Software

Instructions in this workbook assume you will use IBM WebSphere Application Server, Advanced Single Server Edition, (from this point “IBM WebSphere Application Server AEs”) Version 4.0 for Window NT and Windows 2000. You will also use IBM DB2 Personal Developer’s Edition, version 7.2. If you’re using Red Hat Linux, you will find the steps you must follow very similar to, but not exactly the same as the ones you see here. Adjusting the instructions appropriately should not be too difficult.

Downloading IBM WebSphere Application Server AEs

Please make sure your system meets these requirements before downloading the server product:

- ◆ **Memory:** Your computer must have at least 256 Mb of RAM. 512 Mb is recommended.
- ◆ **OS:** The product supports Microsoft Windows NT Server or Workstation with Service Pack 6a or above, or Microsoft Windows 2000. Note that you must use an ID with Administrator authority to install WebSphere.
- ◆ **File System:** You will need approximately 500 Mb of free disk space to install IBM WebSphere Application Server AEs, and an additional 300 Mb to install IBM DB2, Personal Developer’s Edition.
- ◆ **Browser:** In order to use the server’s administration features you must have either Microsoft Internet Explorer 4.01 or higher, or Netscape Navigator 4.7 or higher installed on your computer.

To download IBM WebSphere Application Server AEs, simply point your browser to:

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

Then follow the **Download** link and follow the instructions you find there.

Also note that the product comes with a minimum set of help files in the installation image that you download from the above link. After you download the installation file from that link, you should point your browser to:

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

From this page you can download the IBM WebSphere Application Server AEs documentation set (the InfoCenter). Follow the instructions to download the search-enabled InfoCenter as a .jar file. Be sure to note the directory into which you save the .jar.

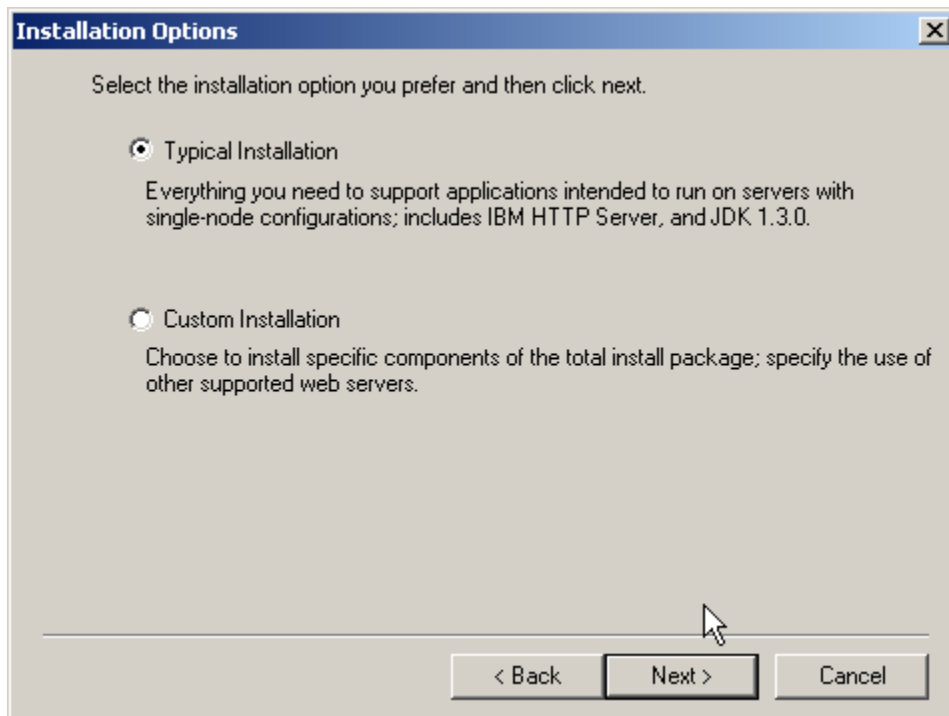
Installing WebSphere AEs on Windows 2000 or Windows NT

Before you begin the installation, ensure that no other programs are running. Installation will not be complete until you restart Windows, either manually or by accepting the installation's offer to restart for you. Also, please note that if you intend to use an HTTP Server with IBM WebSphere other than the IBM HTTP Server included with the software – for instance, Microsoft Internet Information Server (IIS) – then that software must be shut down prior to continuing the installation.

To install the server, you will need to use a freeware or commercial unzip utility to unzip the installation file (*ibmwebas_trial_nt*) to a temporary directory. After unzipping the files to that directory, double-click on the *Setup.exe* file from the Windows Explorer, or from the **Start** menu select **Run...** to start *Setup.exe*. You'll see the customary startup screen for the installation wizard.

After making sure that all necessary applications are shut down, press the **Next** button, which will show the following window:

Figure 1: Installation options

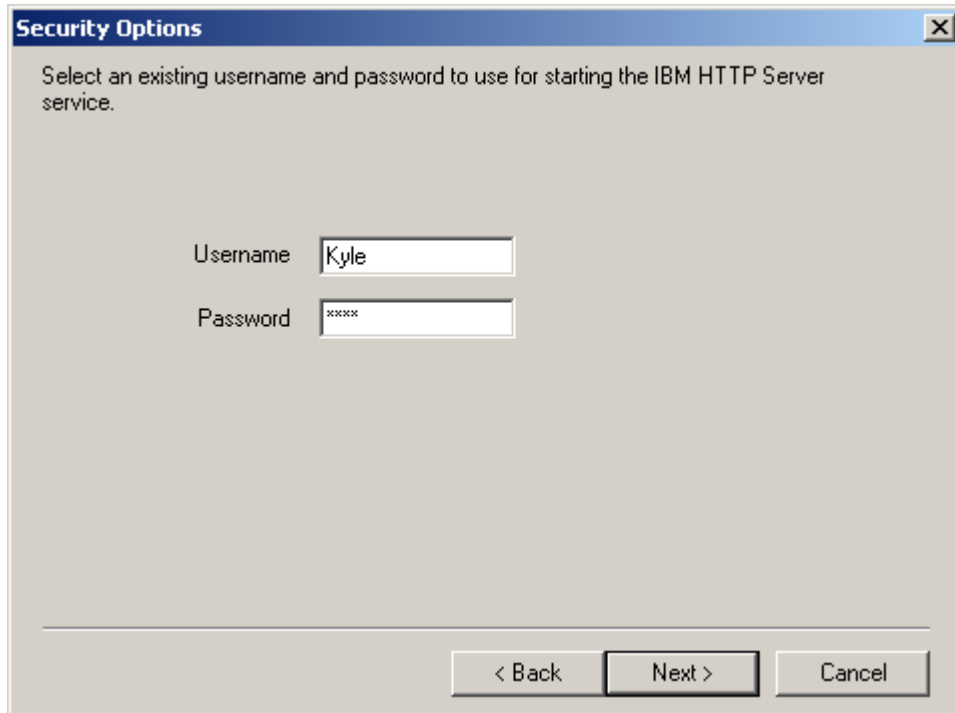


In this workbook we will assume that you do not have another web server installed, and that you can select **Typical Installation**. If you wish to install WebSphere with another web server, then

use **Custom Installation**. If you do, your installation path will differ, but not substantially, from the path we will outline.

Press **Next** to continue to the next screen.

Figure 2: Security options

A screenshot of a Windows-style dialog box titled "Security Options". The dialog has a blue title bar with a close button (X) in the top right corner. The main area is light gray and contains the text "Select an existing username and password to use for starting the IBM HTTP Server service." Below this text are two input fields. The first is labeled "Username" and contains the text "Kyle". The second is labeled "Password" and contains six "x" characters. At the bottom of the dialog, there are three buttons: "< Back", "Next >", and "Cancel". The "Next >" button is highlighted with a black border.

Security Options

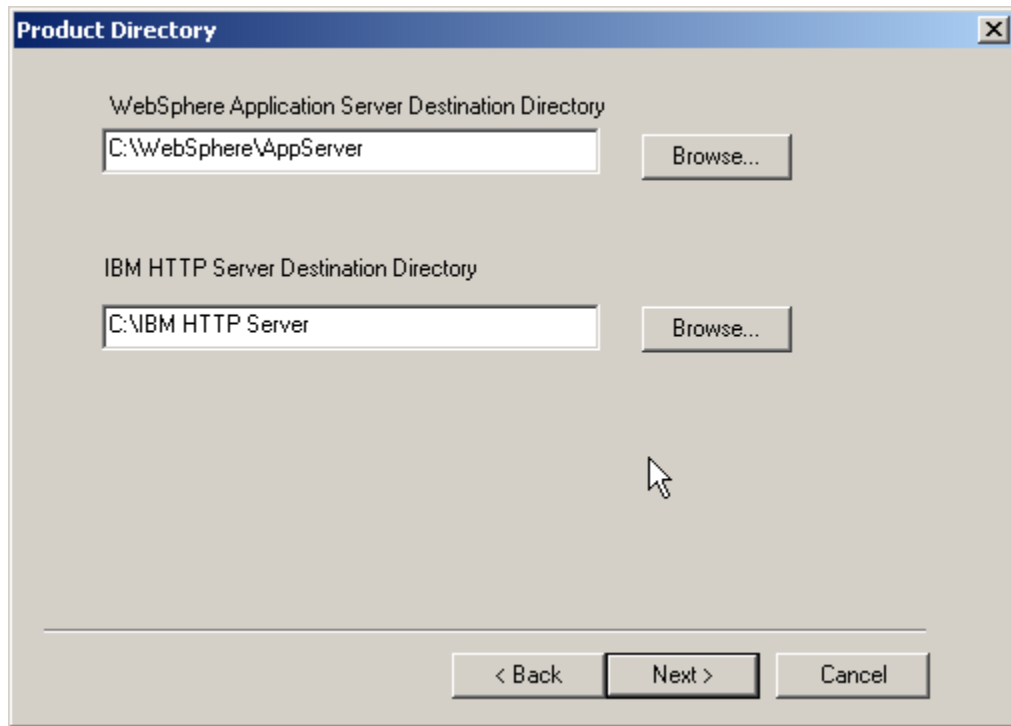
Select an existing username and password to use for starting the IBM HTTP Server service.

Username

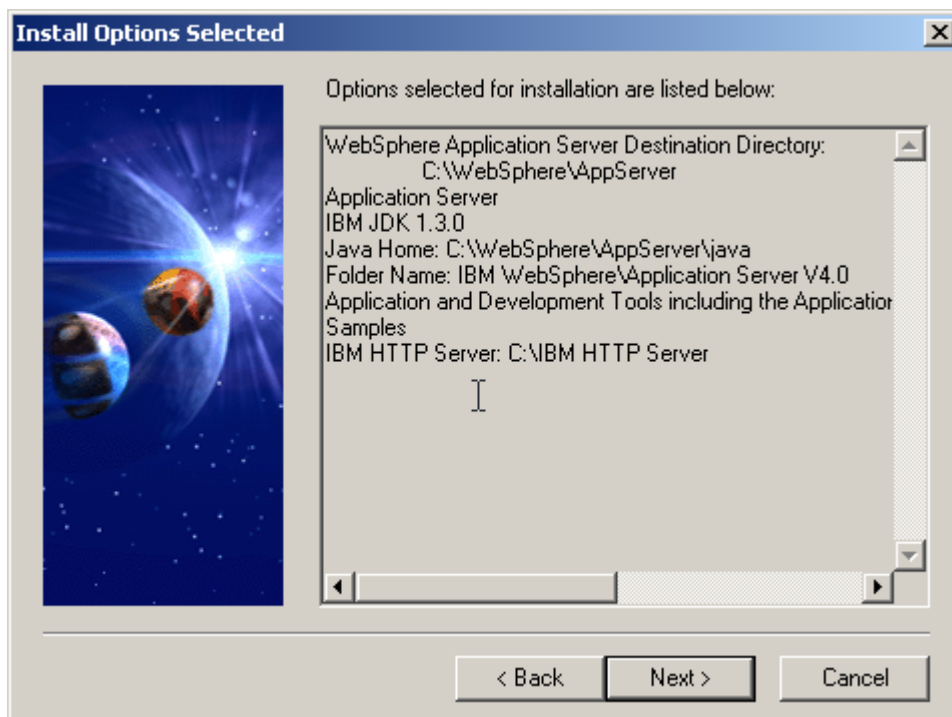
Password

< Back **Next >** Cancel

In this screen you need to enter the username and password for the user ID with administrative privileges on your local machine that you will use to start the IBM HTTP Server service. Enter this information and press **Next**.

Figure 3: Product directories

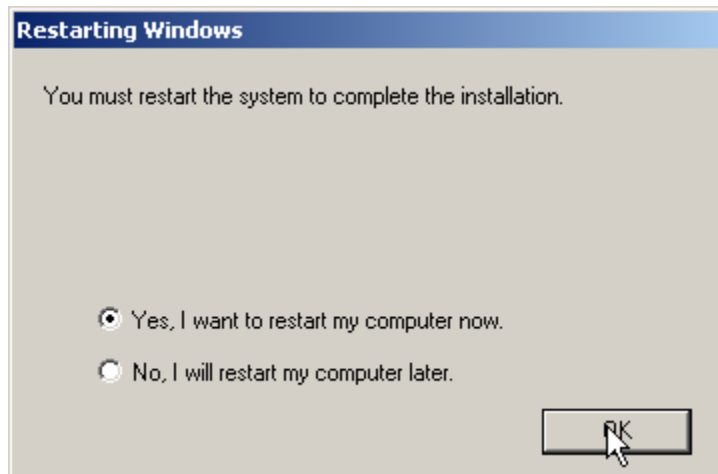
On this screen, you will enter the directories into which you want the WebSphere application server and IBM HTTP Server installed. You can either accept the defaults or choose your own directories. If you choose a new directory, make sure that it has enough space on the file system to install the necessary files. After entering the directories, or taking the defaults, press **Next** to proceed to the next screen. This will provide you with the customary opportunity to select the program folder from the Windows Start Menu to which the icons for IBM WebSphere Application Server will be added. We recommend that you accept the default and press **Next** to move to the following screen:

Figure 4: Installation options selected

This screen simply summarizes the installation options you have selected up to this point. Review these selections carefully, and press either **Back** to change your selections, or **Next** to proceed with the installation.

When you press **Next**, the installation wizard will begin copying files to your hard drive to install the IBM WebSphere Application Server, AEs. You will see a progress bar indicating what is being installed, and a set of icons indicating the amount of hard drive space you have remaining, along with other information. After IBM WebSphere Application Server AEs and the IBM HTTP Server have been installed, a dialog will appear that gives you the option to view the README file or finish immediately. When you press **Finish**, you will see the following dialog:

Figure 5: Restarting Windows



We recommend that you take the default and restart Windows at this point.

Downloading and Installing IBM DB2

One of the strengths of EJBs lies in their ability to access data stored in popular relational databases. In order to use the example programs in this workbook, you will need to install a relational database supported by IBM WebSphere Application Server AEs. The complete list of supported databases can be found at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

This workbook will only provide instructions for using the IBM DB2 Personal Developer's Edition 7.2. This database is a free version (for development purposes only) of the popular and powerful IBM DB2 database. If you want to use another database for the examples, you will have to modify some of the steps given in the instructions on your own. In particular, whenever this workbook provides SQL DDL (Data Definition Language) for table creation, it will be provided only for DB2. The instructions will tell you how to generate the DDL for other databases, but only the DDL for DB2 will be shown in the examples.

To download a free single-user copy for DB2 for development purposes, go to the following URL:

<http://www.software.ibm.com/webapp/download/category.jsp?s=c&cat=data>

On this page, select the link for IBM DB2 Universal Database. Follow that link to download DB2 for Windows Personal Edition 7.2 and download the product installation files using the download method of your choice..

After downloading the product files, you should then be able to follow the instructions on the web pages to install IBM DB2 7.2 Personal Developer's Edition from the product installation files.

Note that this process may require you to reboot your machine several times. It is also important that you install IBM DB2 while logged in with an account that has administrative privileges in Windows NT/2000. Also remember that, while IBM DB2 Personal Developer's Edition has the same memory and OS requirements as IBM WebSphere Application Server, AEs, DB2 requires at least an additional 300 Mb of hard-drive space.

Before You Begin

Before you begin learning how to build EJBs and EJB clients using IBM WebSphere Application Server, AEs, you need to:

- ◆ Learn a little about the exercises in this workbook
- ◆ Set up the DB2 [SAMPLE](#) database
- ◆ Understand the J2EE file packaging structure
- ◆ Get to know the elements of IBM WebSphere AEs, and configure it to use DB2

If you are already familiar with DB2, or have used IBM WebSphere AEs before, you may skip those topics.

Understanding the exercises in this workbook

As you read “the EJB book” (our shorthand for O'Reilly's *Enterprise JavaBeans, 3rd Edition*, by Richard Monson-Haefel, in case you skipped the Preface), you will notice that it contains examples for both the EJB 1.1 and the EJB 2.0 specifications. Because IBM WebSphere Application Server, AEs, complies with EJB 1.1, this workbook will give instructions on how to implement only those examples from the book that the EJB book provides for the EJB 1.1 specification. It is not possible to deploy EJB 2.0-compliant EJBs to WebSphere AEs at this time.

The examples in this workbook are provided in a .zip archive. When you unzip this file to a drive on your machine, notice that its contents expand into the following structure:

```
<Installation Root>
  /Chapter4
    /dev
  /Chapter5
    /Exercise5-1
      /dev
    /Exercise5-2
      /dev
  /Chapter9
    /dev
  /Chapter10
    /dev
  /Chapter12
    /Exercise12-1
      /dev
    /Exercise12-2
      /dev
```

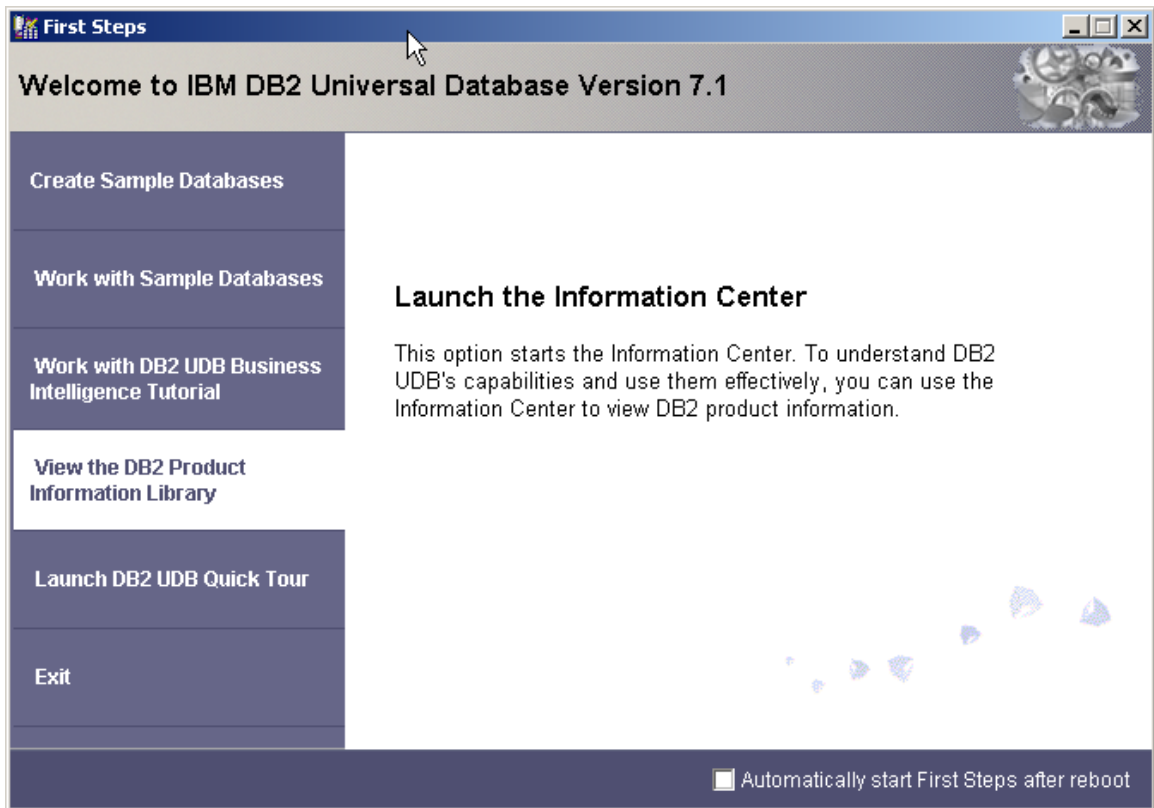
The chapters represented here are the ones from the EJB book that contain exercises for deployment in an EJB 1.1 environment. The lowest-level (*leaf*) directories shown in this structure contain further directories that hold the Java source code (.java files) for the examples provided in the EJB book. They also contain Microsoft Windows batch (.bat) files for compiling these examples and assembling the resulting compiled Java (.class) files into Java Archive (.jar) files.

If you want to try these examples on a platform other than Microsoft Windows 2000 or Windows NT, such as RedHat Linux, then you will need to modify these batch files. The batch files only set environment variables and invoke Java tools like *javac* and *jar* (which are included with IBM WebSphere Application Server, AEs) so it should not prove difficult to convert them to shell scripts that accomplish the same ends.

Setting up the sample database

The first prerequisite is a DB2 database in which you will create your tables for the exercises. The simplest database to use for this is the **SAMPLE** database that DB2 can create automatically. To create **SAMPLE**, from the Windows **Start** Menu choose **IBM DB2>>First Steps**. You will see the following window:

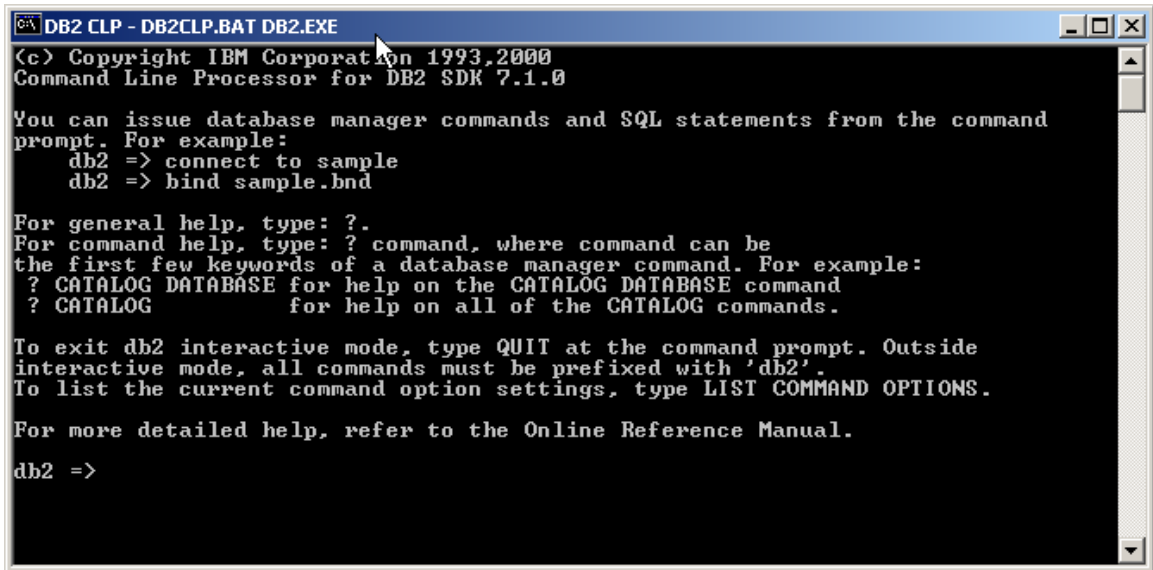
Figure 6: DB2 Universal Database First Steps



Press the **Create Sample Databases** button at the upper right corner of this window, and you will be led through the process of creating the **SAMPLE** database in DB2. When you have finished the process, you can verify that it has worked by invoking the IBM DB2 Command Line Processor (CLP) to execute SQL statements against this database.

To invoke the CLP, select **IBM DB2>>Command Line Processor** from the Windows **Start** menu. The following window will appear:

Figure 7: IBM DB2 CLP



```
DB2 CLP - DB2CLP.BAT DB2.EXE
(c) Copyright IBM Corporation 1993.2000
Command Line Processor for DB2 SDK 7.1.0

You can issue database manager commands and SQL statements from the command
prompt. For example:
  db2 => connect to sample
  db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be
the first few keywords of a database manager command. For example:
? CATALOG DATABASE for help on the CATALOG DATABASE command
? CATALOG           for help on all of the CATALOG commands.

To exit db2 interactive mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the Online Reference Manual.

db2 =>
```

This window allows you to execute SQL commands against any database in DB2. To begin, you must connect to `SAMPLE`. Simply type:

```
connect to SAMPLE
```

at the prompt and press **Enter**. It will respond with the name of the database server it is connecting to, the authorization ID (e.g., your login name in Windows), and the name of the database it is connecting to, `SAMPLE`.

To see whether the database has been correctly populated, type:

```
select * from EMPLOYEE
```

at the prompt and press **Enter**. You should see a number of rows that are part of the sample employee database. In this workbook, you will not use this table (or any of the other tables provided automatically when you created `SAMPLE`); instead, you will create your own tables. Performing this `select` operation does, however, demonstrate to you that the database is up and functioning, and shows you how to execute arbitrary SQL statements against the `SAMPLE` database in DB2.

It is well worth your while to learn a little more about the CLP, as you will use it extensively in this workbook to:

- ◆ create tables
- ◆ verify that tables have been populated
- ◆ delete rows from tables

You can read about it in the DB2 Information Center that was installed with DB2, or you can simply type **?** and press **Enter** to read the help screens for the CLP.

Setting DB2 to Use JDBC 2.0

A final DB2 configuration change that you must make in order to use DB2 with WebSphere is that you must set it up to provide Support for the JDBC 2.0 API. By default DB2 supports only the JDBC 1.1 API. However, J2EE 1.2 and EJB 1.1 require the use of the JDBC 2.0 API. Luckily, configuring this support is a trivial matter. Simply open a Windows command prompt and `cd` to the directory where you installed DB2 (usually *C:\Program Files\SQLLIB*). At the command prompt type:

```
usejdbc2.bat
```

and press the **Enter** key. This batch file changes the default support for JDBC to JDBC 2.0.

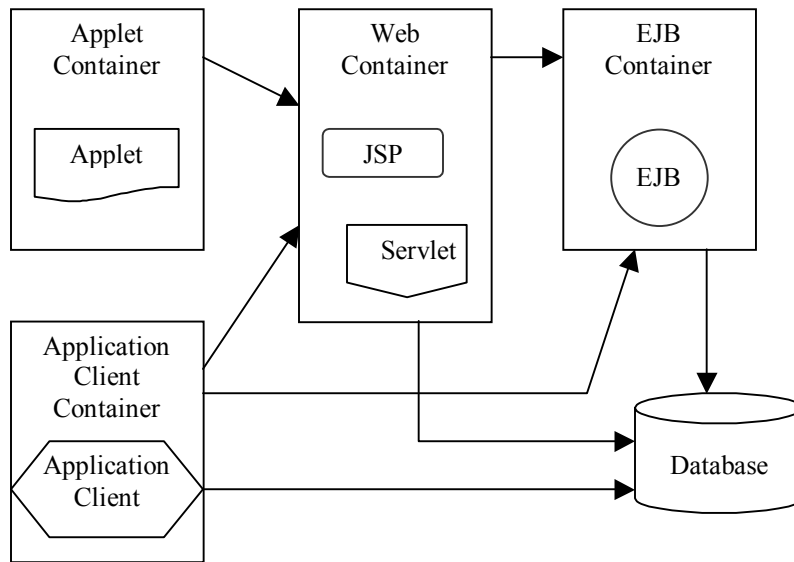
An Introduction to J2EE and IBM WebSphere AEs

Now that all of the necessary software is installed on your system, you are ready to begin learning how IBM WebSphere Application Server, AEs, works. First you will need to supplement the material in the EJB book with some additional information on J2EE (Java 2 Platform, Enterprise Edition).

Enterprise JavaBeans is just one a number of technologies that compose the J2EE platform. Others include:

- ◆ Servlets – a technology for writing server-side web scripts similar to CGIs
- ◆ JavaServer Pages – a template technology for embedding dynamic server-side Java code in HTML or XML pages
- ◆ Java Message Service – a standard API for communicating with Message Oriented Middleware systems
- ◆ J2EE Connectors -- a standard API and specification for communicating with Enterprise Information Systems

One of the key organizing principles of J2EE is that it is composed of two fundamental types of objects: *containers* and *components*. The following diagram, adapted from the J2EE Specification, shows the kinds of containers present in a J2EE application environment, and the components that are deployed in those containers:

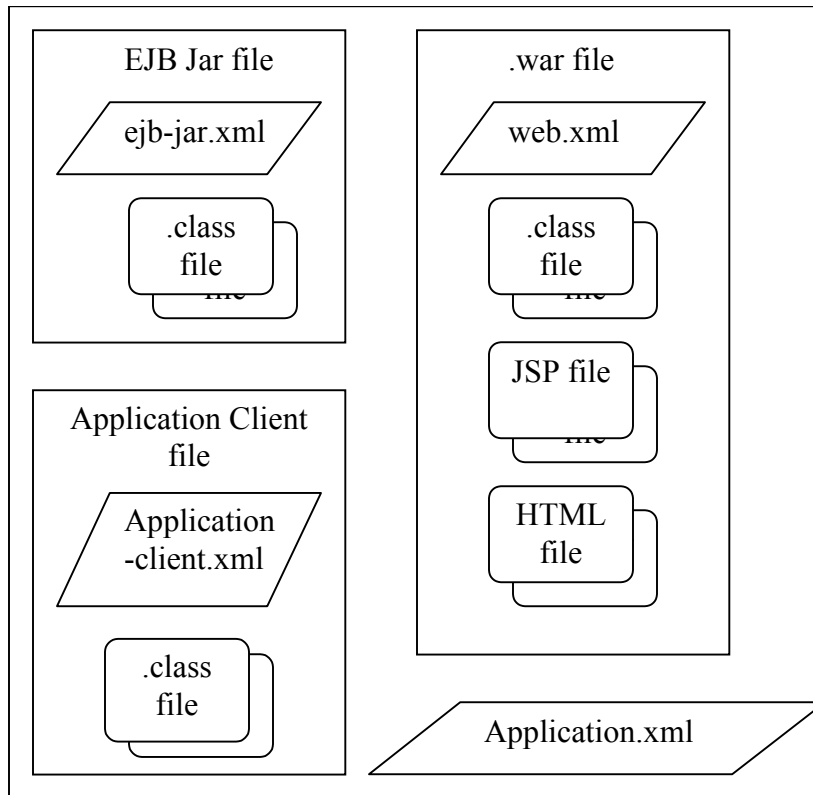
Figure 8: Containers and components

In J2EE the various components are linked together into logical units through a nested set of special *.jar* files and deployment descriptors. Chapter 2 of the EJB book describes how EJBs are deployed as a set of compiled java bytecode files (*.class* files) and a deployment descriptor (an *.xml* file) to form an EJB *.jar* file. This same approach of grouping files with an XML deployment descriptor is also used to form other basic structures in J2EE. Besides EJB *.jar* files, the basic building blocks of J2EE are:

- ♦ *Web archive (.war)* files, consisting of a set of *.class* files that may be Java servlets or classes that servlets need to carry out their functions, JavaServer Pages (*.jsp* files), and other web resources, such as *.html* and *.gif* files. Web Archive files must also contain an XML deployment descriptor (always named *web.xml*) that describes their structure.
- ♦ *Application client .jar* files, consisting of a set of *.class* files that make up a Java application, including one class whose *main* method starts the application and any resource files (e.g., property files) it needs. An application client *.jar* file will also contain an XML deployment descriptor that describes the application.

EJB *.jar* files, web archive files, and application client *.jar* files can be combined to form a final top level of application structure called an *enterprise archive (.ear)* file. An *.ear* file consists of any combination of any number of files of the above three *.jar* types, plus an additional XML deployment descriptor for the combination.

The following diagram shows how the entire *.ear* file structure works.

Figure 9: Enterprise archive deployment structure

So, what bearing does this review of J2EE packaging have on learning how to deploy EJBs in IBM WebSphere Application Server AEs? Well, you have to understand the packaging conventions because, as a J2EE-compliant application server, WebSphere will not permit you to deploy applications, be they single EJBs or EJB client applications or larger applications with all of the component types, unless you package them according to the J2EE deployment rules. If you understand this restriction, then the following description of the components of IBM WebSphere Application Server AEs will make much more sense.

IBM WebSphere AEs Elements

- Note: If you are already familiar with the components of WebSphere Application Server AEs, and have already finished configuring it to use DB2, you may skip this section and proceed to the exercises.

IBM designed its J2EE 1.2-compliant WebSphere Application Server AEs for deployment of applications that require only a single application server machine, and for development of applications for single- **and** multiple-machine environments. Another version of the product, IBM WebSphere Application Server, Advanced Edition, supports applications deployed over multiple application server machines for performance, fail-safety, and scalability. We will not cover those particular features in this workbook. For information on them, refer to the product documentation on the IBM website.

When you install the server product, you will find the following set of components has been installed. To reach them from the Windows desktop, select **Start→IBM WebSphere→Application Server 4.0**.

- ◆ The *Application Assembly Tool (AAT)* is a development and deployment tool for building EJB *jar* files, application client files, and web archive files, and for assembling these into enterprise archive (*.ear*) files.
- ◆ The *Application Server* is a Java process that hosts 0..1 web containers and 0..1 EJB containers. The application server also contains administrative objects that are modified by the web-based *Administrative Console*, which you will also use to deploy J2EE applications into the Application Server.
- ◆ *First Steps* is a small application that will help you to start and stop the application server, and to launch the Administrative Console.

You can begin your exploration of WebSphere AEs (and prepare for the exercises that follow) by starting First Steps. From the Windows **Start** menu, select **IBM WebSphere→Application Server V4.0→First Steps**. When the application is fully started, you will see the following screen:

Figure 10: First Steps menu



You can reach all the tools you'll need (the Application Server, The Administrative Console, and the Application Assembly Tool) from this screen. One thing to keep in mind is that the Administrative Console is dependent on the Application Server. You can start the console only if the server is currently running, because the web container that hosts the HTML and JSP pages that make up the console runs within the server. If you ever try to start the Administrative Console and your browser reports "localhost was not found," either you have not started the Application Server, or you have not given it enough time to complete its startup.

If you would like to see how this works, begin by pressing the **Start the Application Server** button. After a short time you will see a console window appear that will give you informative messages about how the startup of the application server is progressing:

Figure 11: Console with startup messages

```

E:\WINNT\system32\cmd.exe

WebSphere Application Server, Advanced Single Server Edition V4.0
Application Server Launcher
Copyright (C) IBM Corporation, 2001

The configuration file was defaulted to:
I:\WebSphere\AppServer\config\server-cfg.xml
Using the single available node or the localhost node.
Using the single available server.
Initiating server launch.
Loaded domain "WebSphere Administrative Domain".
Selected node "kgbrown1".
Selected server "Default Server".
WSPL0065I: Initiated server launch with process id 2036.
Time mark: Sunday, September 9, 2001 10:55:35 PM EDT
Waiting for the server to be initialized.
Time mark: Sunday, September 9, 2001 10:55:44 PM EDT
Initialized server.
Waiting for applications to be started.
Time mark: Sunday, September 9, 2001 10:56:23 PM EDT
Started applications.
WSPL0057I: The server Default Server is open for e-business.
Please review the server log files for additional information.
Standard output: I:\WebSphere\AppServer\logs\default_server_stdout.log
Standard error: I:\WebSphere\AppServer\logs\default_server_stderr.log

```

There is an important point that you need to become familiar with here. The console will identify the log files the application server places its informational messages in. The defaults, shown here, are

<Install Drive>\WebSphere\AppServer\logs\default_server_stdout.log

and

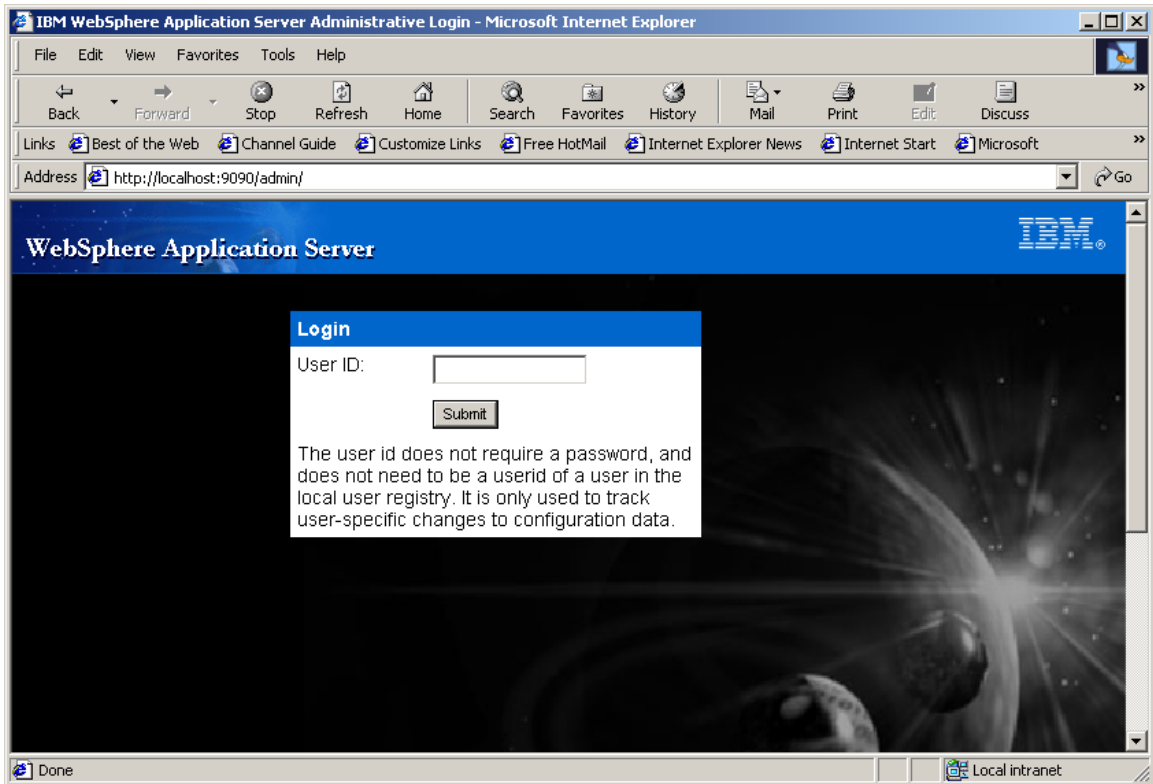
<Install Drive>\WebSphere\AppServer\logs\default_server_stderr.log

To find out whether the application server started correctly, look for the following message:

```
||| WSPL0057I: The server Default Server is open for e-business
```

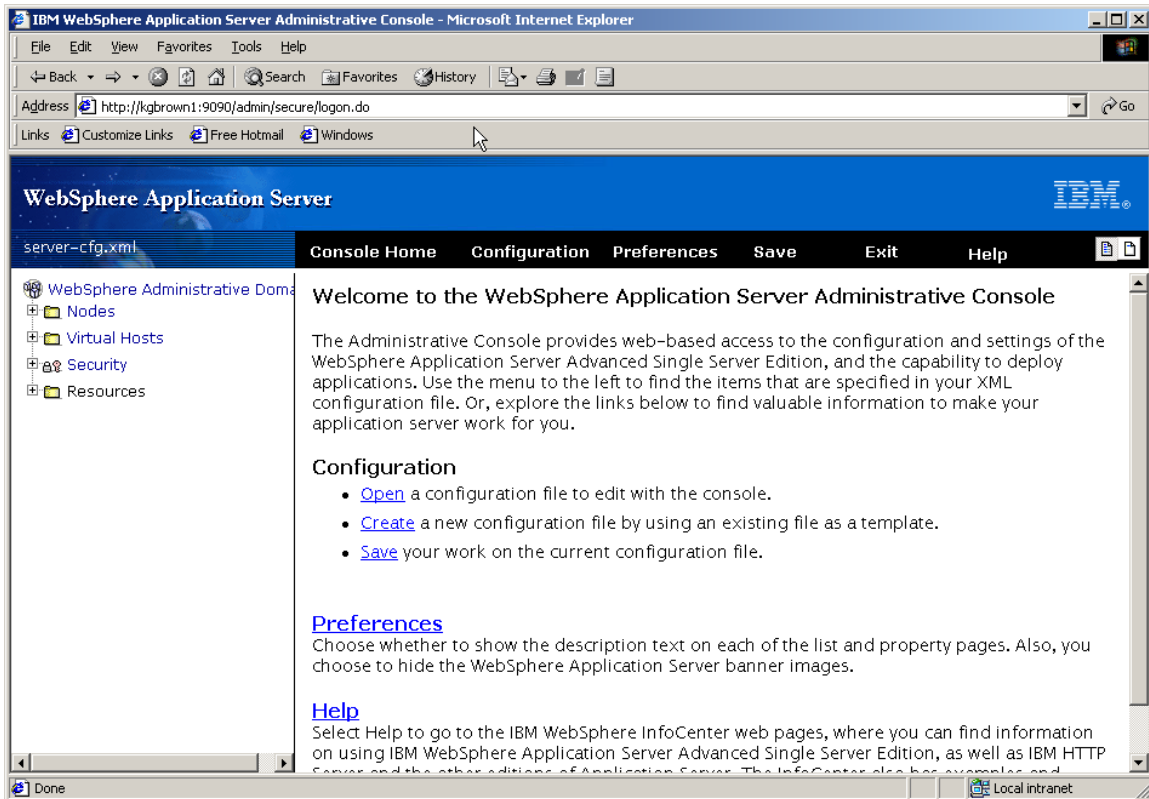
If you see this message then startup was successful. Even then, however, it is worth your while to scan backward through the standard output and standard error files to see if any part of the startup sequence failed. Note that it may take a while for this message to appear, so be sure that the sizes of the log files have remained constant for a minute or so before you look for additional messages.

Now that you've started the application server, you can start the Administrative Console. From the First Steps application, choose **Launch the Administrative Console**. If everything works correctly, you should see a login screen for the Administrative Console open in a browser:

Figure 12: Administrative Console login

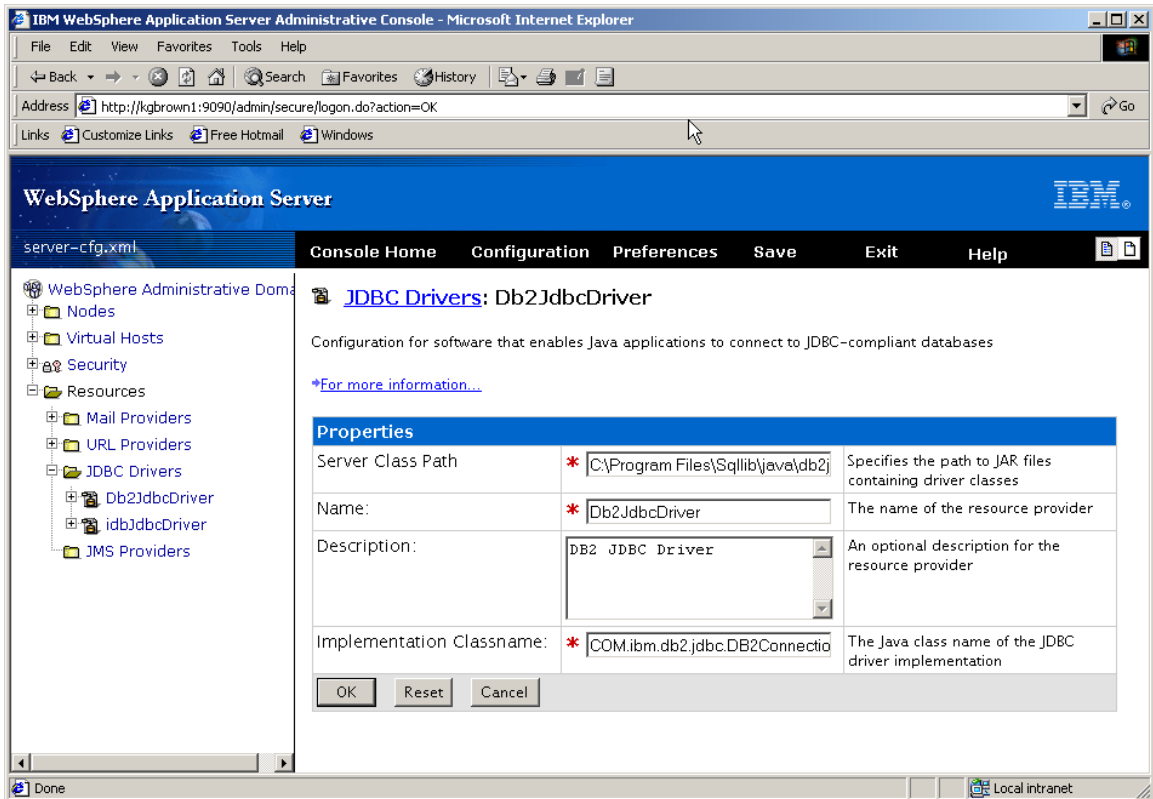
Type in any ID here (your Windows ID works fine, although anything will do, as the screen text indicates). Then press the **Submit** button to reach the following page:

Figure 13: Administrative Console – first startup



While it appears that everything is functioning, you do have a step to complete in order to use DB2 with WebSphere: You haven't yet told WebSphere where DB2 was installed, and you cannot use the DB2 JDBC driver with WebSphere until you do. Begin addressing that problem by clicking on the + sign next to the *Resources* folder to expand the folder. Next, click on the + sign next to *JDBC Drivers* to expand the list of available JDBC drivers. Then click on the *Db2JdbcDriver* link under *JDBC Drivers* to bring up the following screen:

Figure 14: DB2 JDBC driver configuration

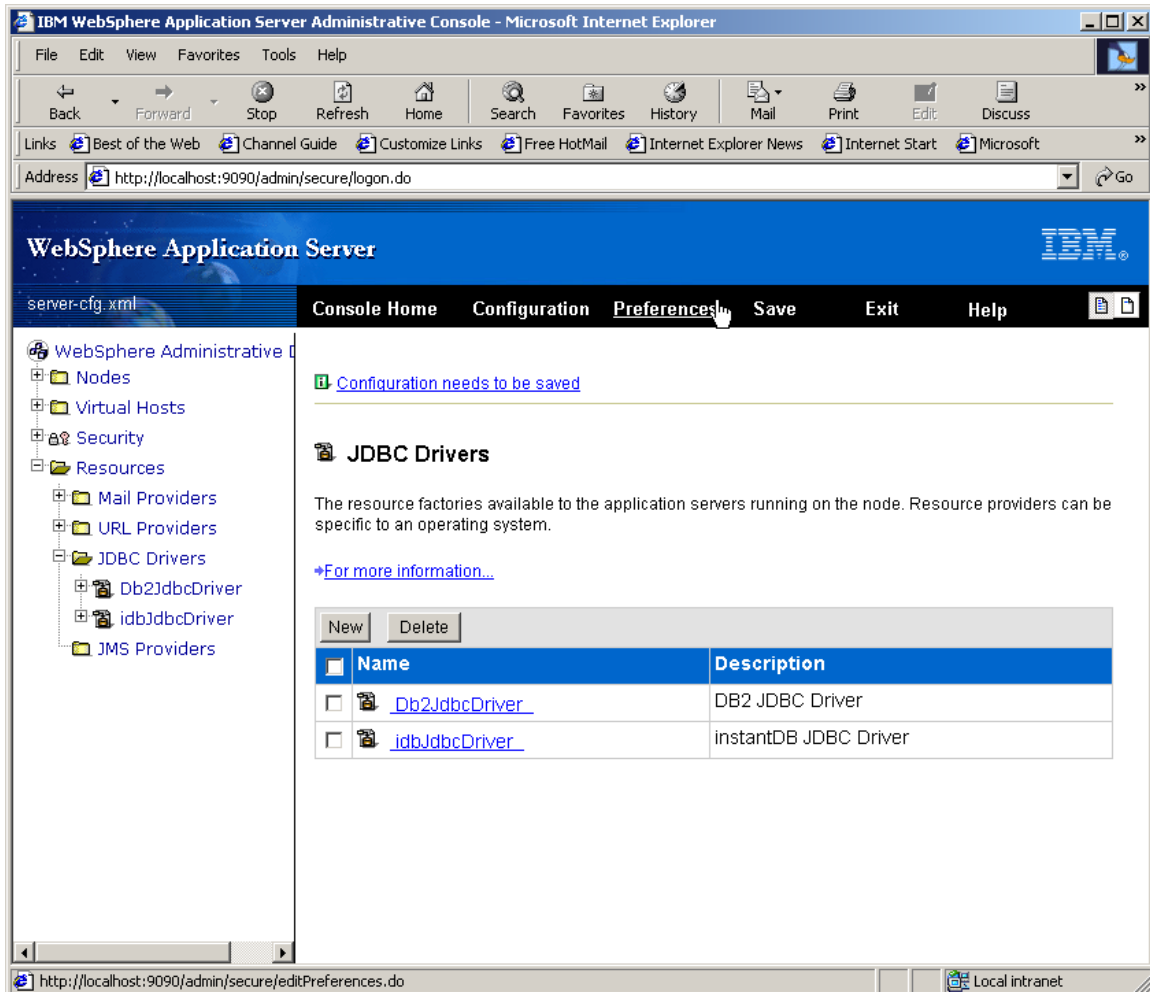


Now you can fix the problem by typing in the path to the *db2java.zip* file that is part of the DB2 Personal Edition installation. By default this is found in...

C:\Program Files\SQLLIB\java\db2java.zip

...but if you changed the path to DB2 when you installed it, you may have to locate this file yourself. In any case, type in the location of this file in the **Server Class Path** text field and press the **OK** button. You will then see the following page:

Figure 15: JDBC driver installation updated

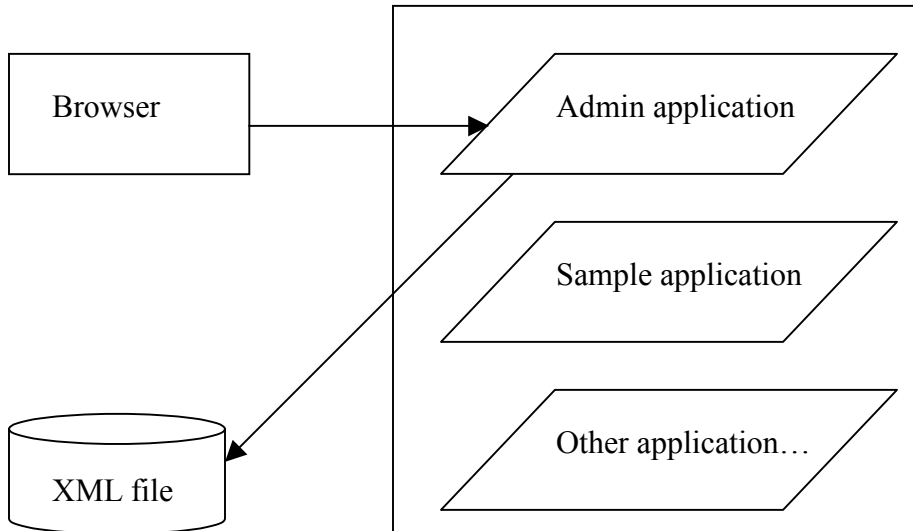


This shows you that the JDBC driver has been updated - but not yet permanently. The information icon at the top of the frame indicates that you need to save this configuration. To understand why a save is necessary, you must take a short detour into the way that IBM WebSphere Application Server, AEs, is structured:

By default, a single JVM handles both the administration of the applications deployed into WebSphere AEs and their execution. The administration tasks are handled by an Administration application that runs within the JVM that contains the other deployed J2EE applications. As you change configuration elements in the administration web pages, the admin application modifies

server-cfg.xml, a configuration file on the local drive where you installed WebSphere AEs. The following diagram illustrates how this process works:

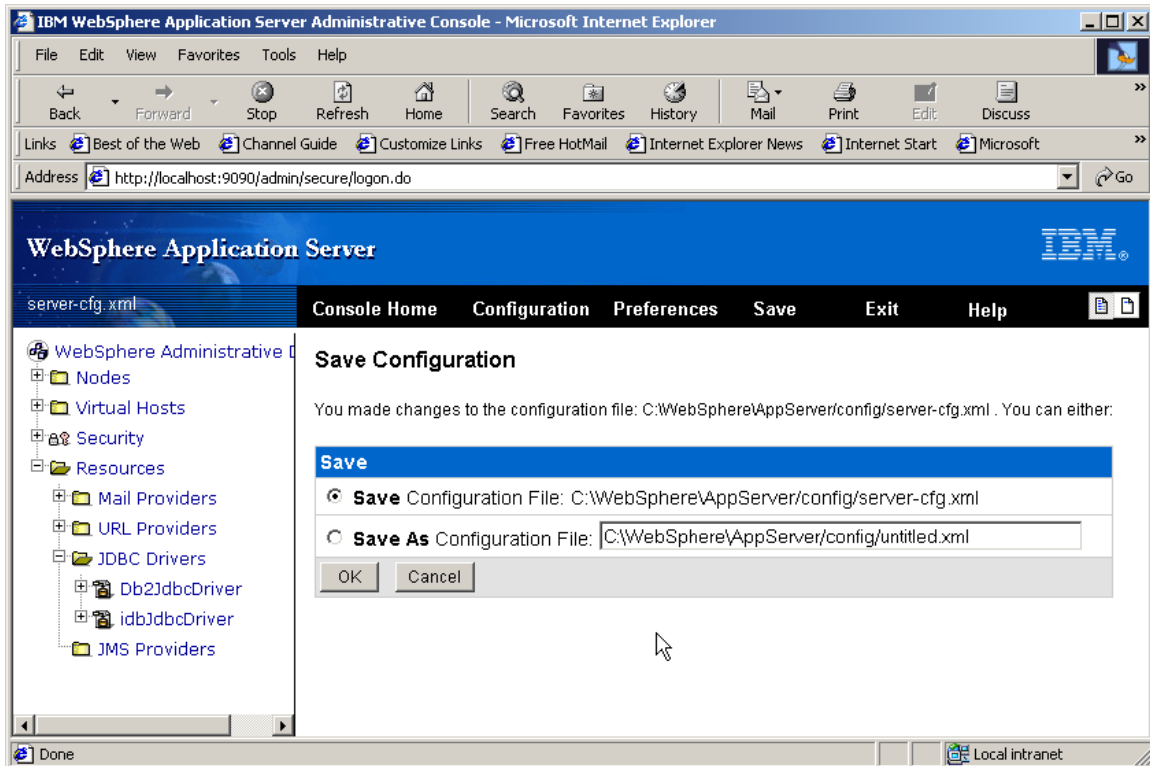
Figure 16: Structure of WebSphere AEs



The admin program uses the XML file at JVM startup, to determine how applications should be loaded into the JVM. It will be able to do so properly only if you tell it to save the configuration – i.e., save the XML file – after every change you make to any administrative object.

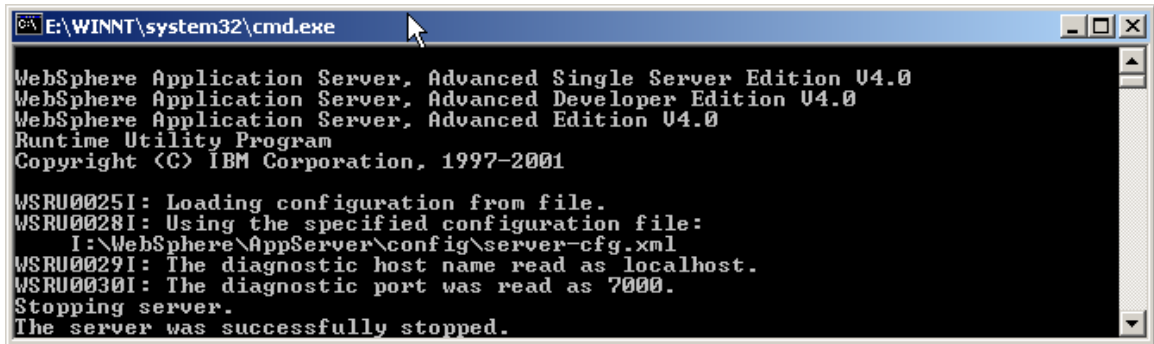
End of detour! To get back to configuring Websphere AEs, press the **Configuration needs to be saved** link. You will see the following screen:

Figure 17: Saving your configuration



At this point, simply press **OK** to save the configuration in *server-cfg.xml* file. Remember: these changes will not take effect until the next time the server is started... which leads you to your final task, shutting down the server. First close the browser that is showing the administrative console. Then go back to the First Steps window and choose **Stop the Application Server**. The following console window will indicate that the application server has stopped:

Figure 18: Console with application server stopped



```
E:\WINNT\system32\cmd.exe
WebSphere Application Server, Advanced Single Server Edition U4.0
WebSphere Application Server, Advanced Developer Edition U4.0
WebSphere Application Server, Advanced Edition U4.0
Runtime Utility Program
Copyright (C) IBM Corporation, 1997-2001
WSRU0025I: Loading configuration from file.
WSRU0028I: Using the specified configuration file:
I:\WebSphere\AppServer\config\server-cfg.xml
WSRU0029I: The diagnostic host name read as localhost.
WSRU0030I: The diagnostic port was read as 7000.
Stopping server.
The server was successfully stopped.
```


Exercises for Chapter 4

Exercise 4.1: A Simple Entity Bean

In this exercise you will create the Cabin EJB and deploy it into IBM WebSphere Application Server. You will also create and run the *Client_1* and *Client_2* programs, whose listings appear in the EJB book.

To accomplish these tasks, you will:

1. Compile the Java source code for the EJB and the client programs into *.class* files
2. Create an EJB *.jar* file for the Cabin bean
3. Use the *Application Assembly Tool (AAT)* to assemble Java *application client .jar* files for *Client_1* and *Client_2*
4. Use the AAT to assemble the EJB *.jar* file and the two application client *.jar* files into an *enterprise archive (.ear)* file
5. Use the AAT to generate deployment code for the *.ear* file
6. Use the DB2 Command Line Processor (CLP) to create a table for the Cabin bean in the DB2 [SAMPLE](#) database.
7. Use WebSphere's Administrative Console to create a DB2 sample data source (*jdbc/SampleDB*) attached to the [SAMPLE](#) DB2 database
8. Use WebSphere's Administrative Console to deploy the new *.ear* file
9. Test your clients with the *launchClient* tool in WebSphere AEs

Step 1: Compile the Java source code

You will begin by compiling the Java source code for the example you are going to use. You can use an IDE to do this as the EJB book suggests; but to make these instructions available to the widest possible audience, we have also provided a Windows batch file that can do the job as well:

```
<InstallDrive>/EJBWorkbook/Source/Chapter4/dev/CabinBeanCompile.bat
```

If you edit this file you will see that it takes advantage of the fact that, when you install WebSphere, you also install a copy of the IBM JDK 1.3. An environment variable named [WAS_HOME](#) refers to the installation directory for WebSphere. The JDK is installed in the *java* directory below [WAS_HOME](#).

Execute the batch file by opening a command prompt, changing your current directory to the above directory, and typing [CabinBeanCompile](#) at the prompt. After the batch file completes its execution, verify that you now have *.class* files for all of the *.java* files in the *com/titan/cabin* directory.

Step 2: Create an EJB .jar file for the Cabin bean

As you read in the EJB book, an EJB *.jar* file is simply a *.jar* file that contains both the *.class* files for the home and remote interfaces and the bean implementation class for your EJB, as well as an XML deployment descriptor. Remember also that the XML deployment descriptor must be placed in a directory in the *.jar* file named *META-INF*, and must be named *ejb-jar.xml*.

The *dev* directory contains a *META-INF* directory that in turn contains the *ejb-jar.xml* file described in the EJB book. Executing the *BuildCabinJarFile* batch file will invoke the *jar* tool in the JDK and build this *.jar* file. Do that now: Type `BuildCabinJarFile` at the command prompt, and when it's done verify that a *.jar* file named *cabin.jar* containing all the above information has been created.

Later, you will learn an alternate way to build EJB *.jar* files that uses WebSphere's Application Assembly Tool.

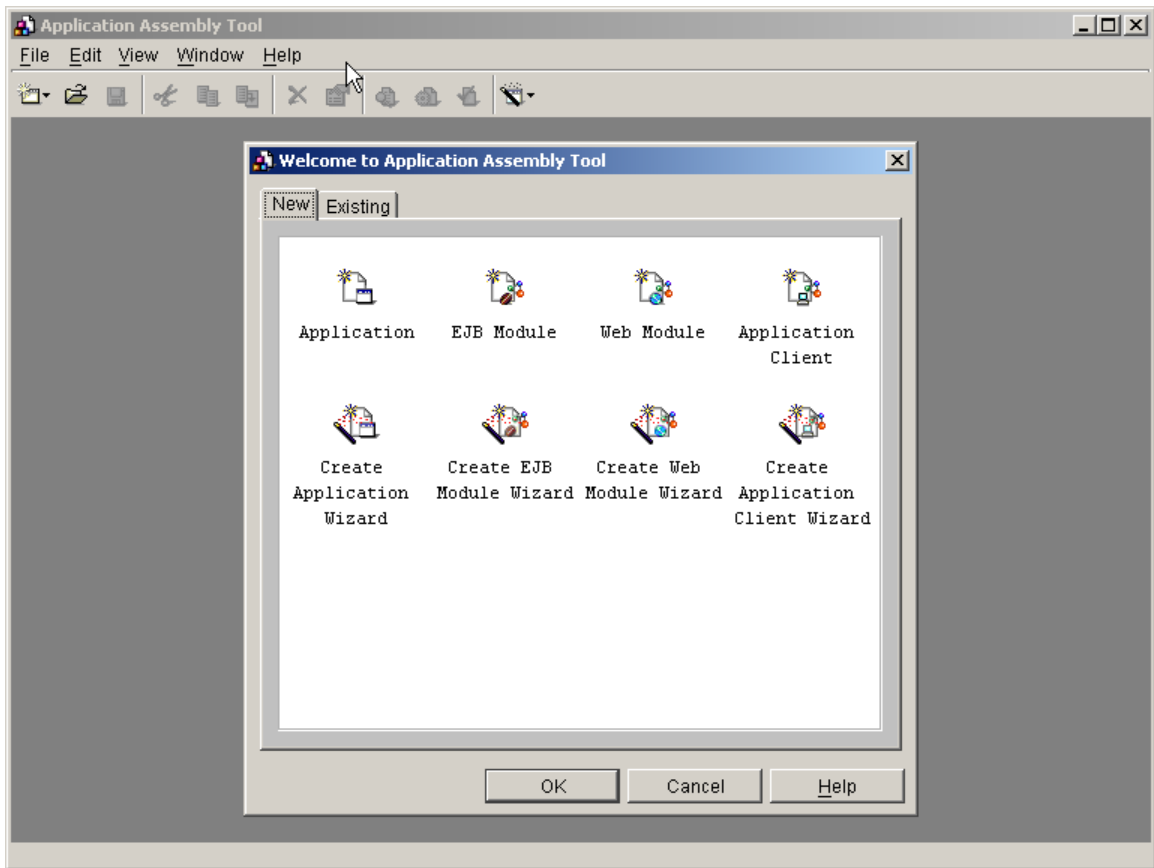
Step 3: Create application client .jar files

Your first use of the AAT, however, will be to build client *.jar* files. The Application Assembly Tool is a Java application that helps you assemble application client *.jar* files, EJB *.jar* files, web archive files, and enterprise archive files.

To begin, make sure that the First Steps application is up and running. If it is not, start it according to the instructions provided earlier in the workbook. Then, in First Steps, start the AAT by selecting the option **Launch the Application Assembly Tool**.

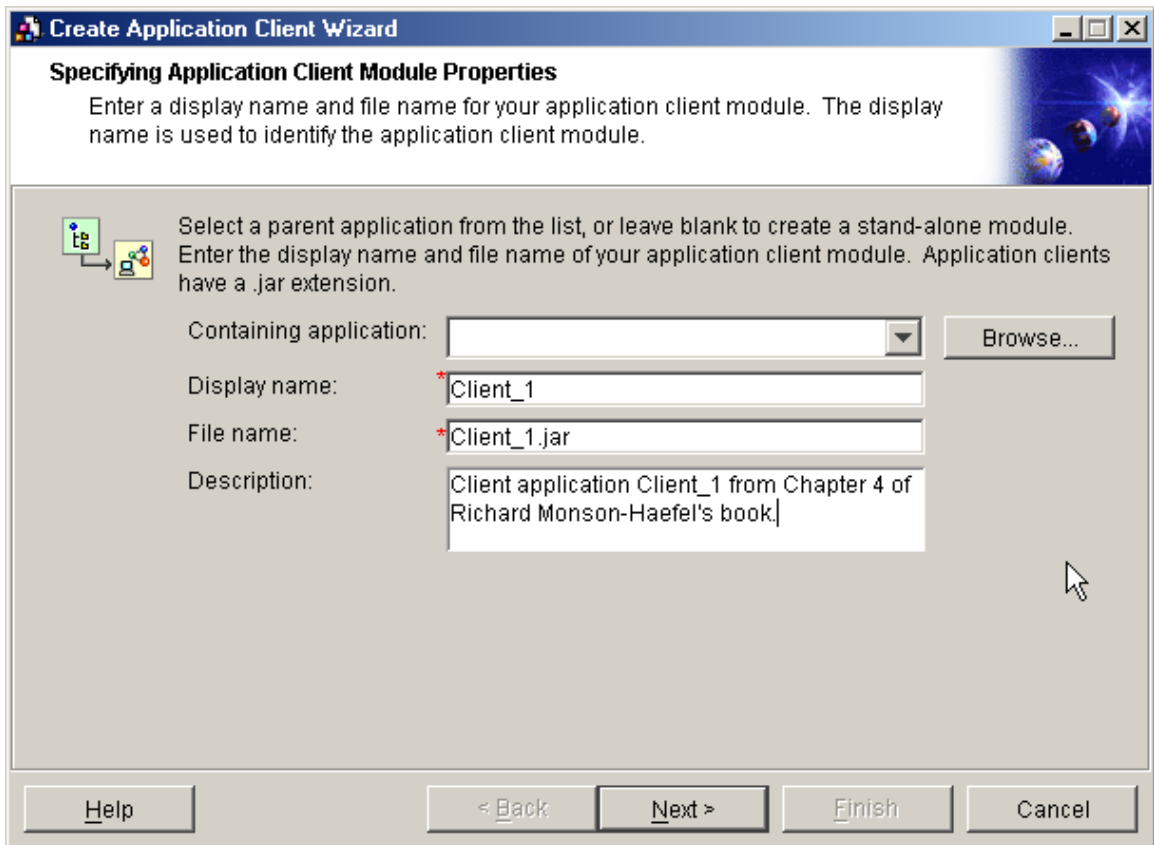
When you start the AAT, you will see the following screen:

Figure 19: Application Assembly Tool – start screen



Begin building the first of your application client *.jar* files by selecting the **Create Application Client Wizard** icon and pressing **OK**. The wizard will display the module properties:

Figure 20: Specifying module properties



Create Application Client Wizard

Specifying Application Client Module Properties

Enter a display name and file name for your application client module. The display name is used to identify the application client module.

Select a parent application from the list, or leave blank to create a stand-alone module. Enter the display name and file name of your application client module. Application clients have a .jar extension.

Containing application: Browse...

Display name: *Client_1

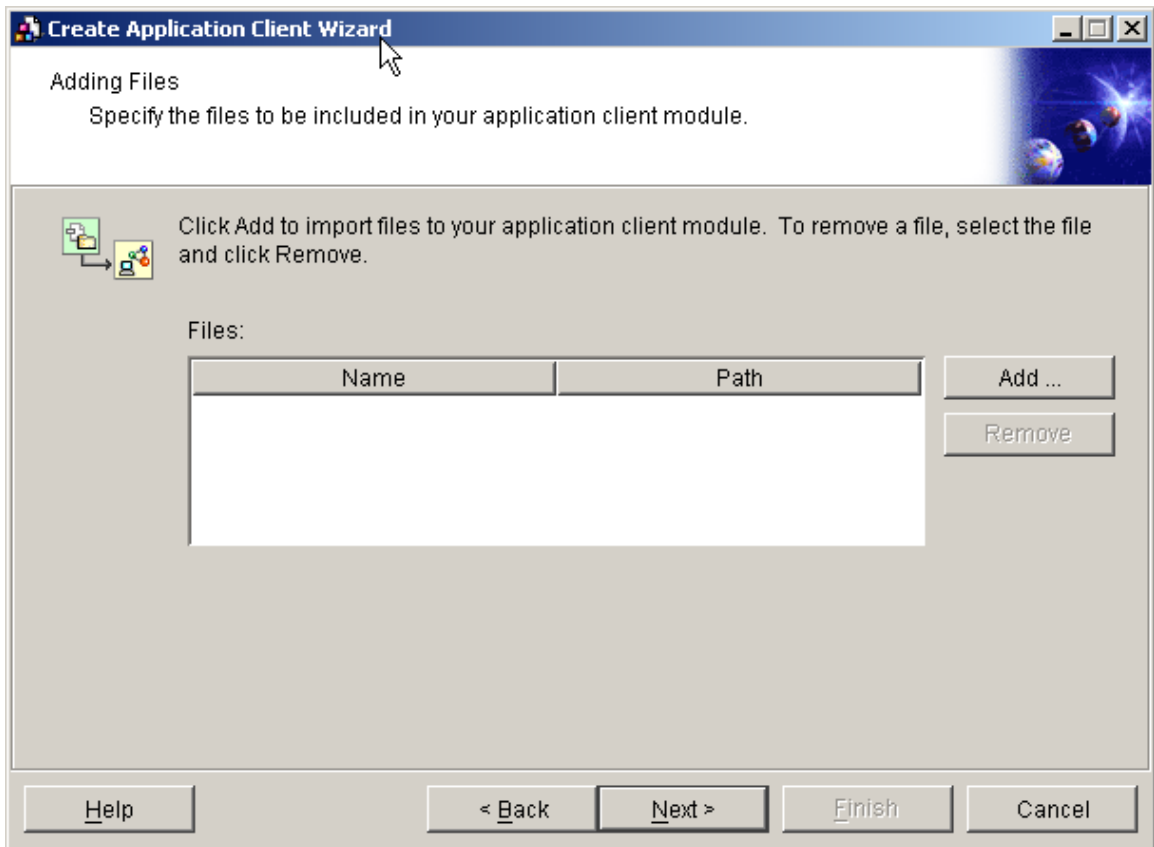
File name: *Client_1.jar

Description: Client application Client_1 from Chapter 4 of Richard Monson-Haefel's book

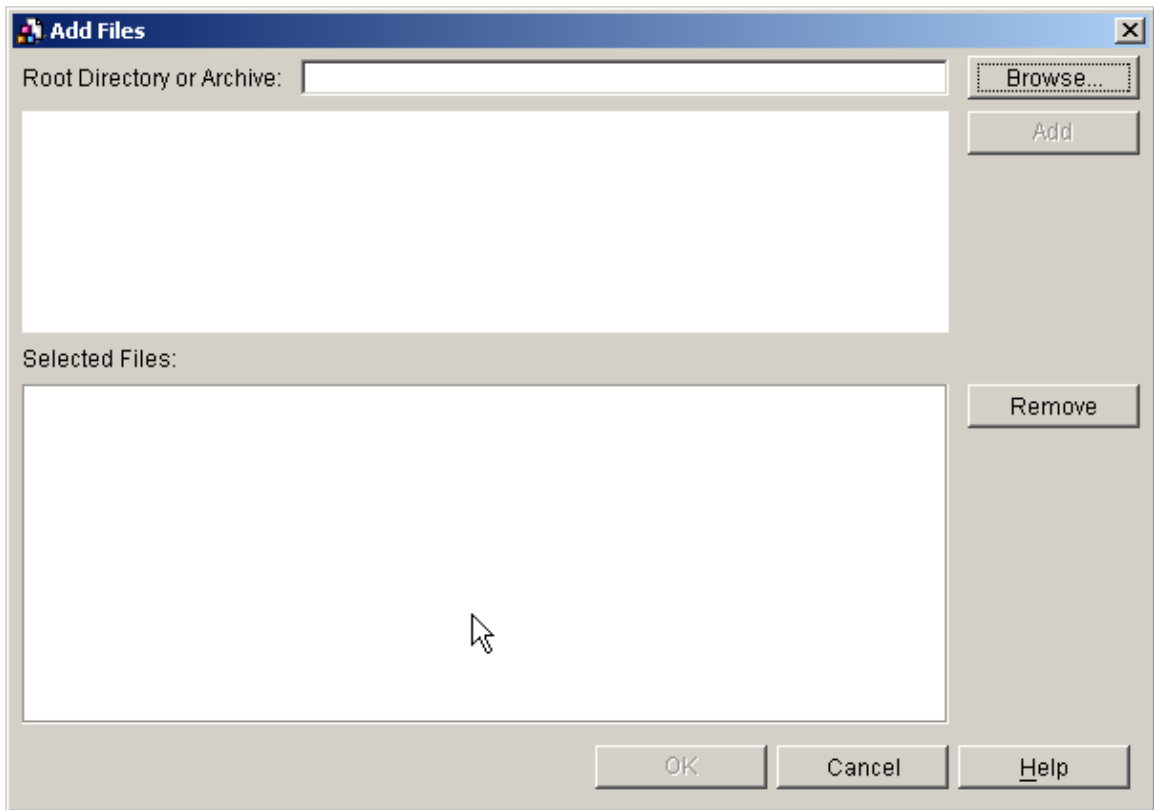
Help < Back Next > Finish Cancel

In this screen, set the client *.jar* file's **Display name** (the name that will be shown when you add this to your *.ear* file later) to be **Client_1**, and set the **File name** to be **Client_1.jar**. You can set the description to be anything you like. Leave the **Containing application** field blank – you will be creating the *.ear* file later. Press the **Next** button to move to the next screen:

Figure 21: Adding files

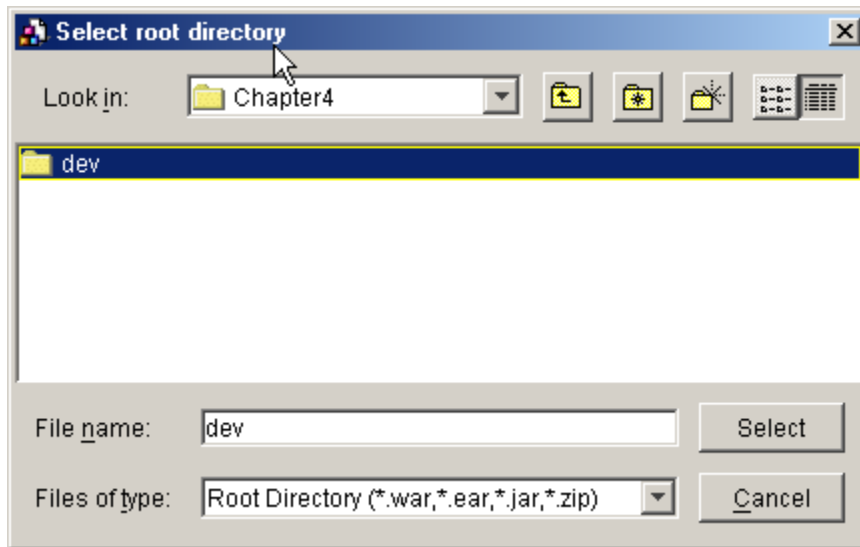


This screen allows you to add files to your application client. You will normally add the file that contains the Java class with the `main` method that starts your application, as well as any other `.class` files your application requires. In this case, you will add only the single `Client_1.class` file described in the EJB book. To do so, press the **Add...** button, which will bring up this screen:

Figure 22: Selecting files to add

This dialog allows you to add files to the *jar* file. You can either press the **Browse...** button, which allows you to select a path for the root directory, or type the root directory name directly. In your case, press **Browse...**, which will bring up a standard Windows file selection dialog:

Figure 23: Selecting the root directory



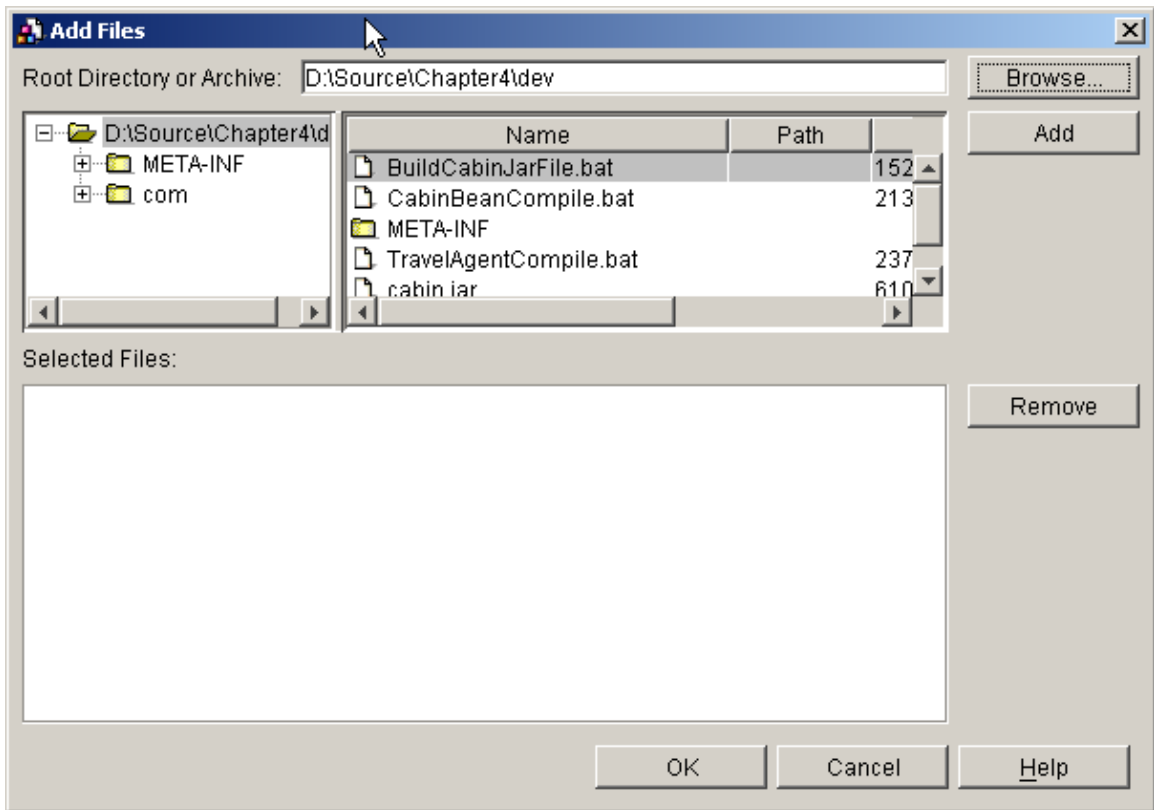
Performing this step correctly is very important. Use this dialog to navigate down to the directory in which you installed the source code for the workbook. Remember that the source code is in:

<InstallDrive>/EJBWorkbook/Source/Chapter4/dev/

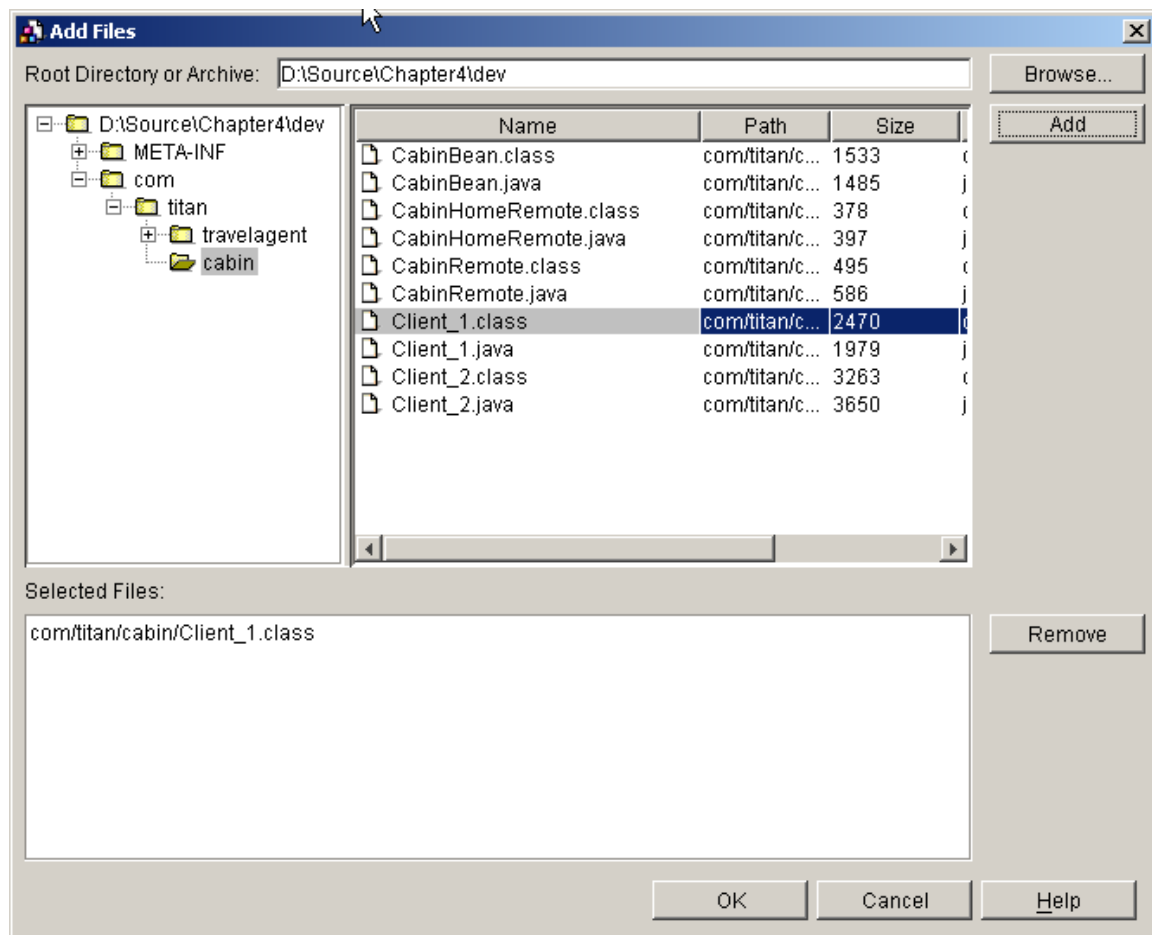
Double-click directory names until you reach the *Chapter4* directory, and highlight the *dev* directory, as in the above diagram, then press **Select**. If you choose the wrong directory here, the path to your class files will not match their fully-qualified Java package names – and the mismatch will confuse the WebSphere class loaders.

After you press **Select**, the Add Files screen will look like this:

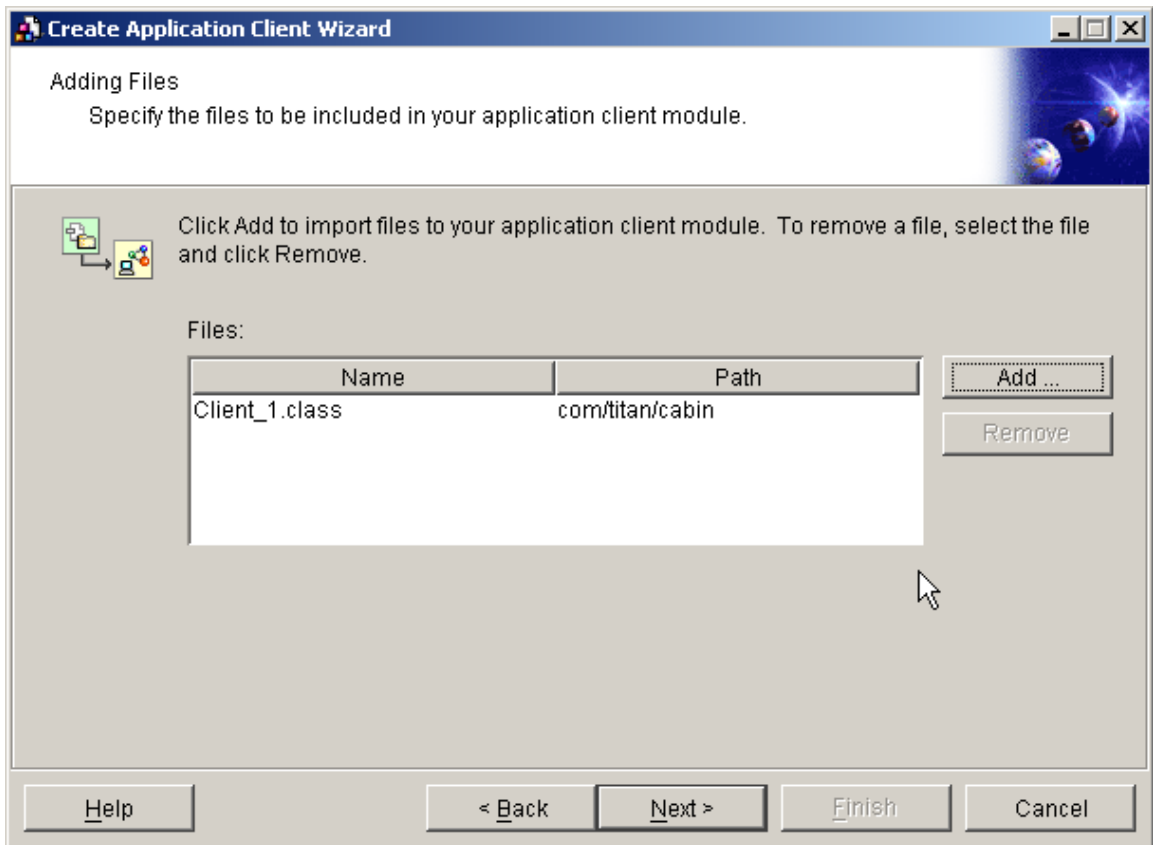
Figure 24: Adding files – showing the dev directory



Your installation drive may be different from that shown above, but you should still see the *com* and *META-INF* directories. Click on the + signs to open the *com/titan/cabin* subdirectory. In this subdirectory, you will see the *Client_1.class* file. Highlight that file and press **Add**. The Add Files dialog should then look like the following:

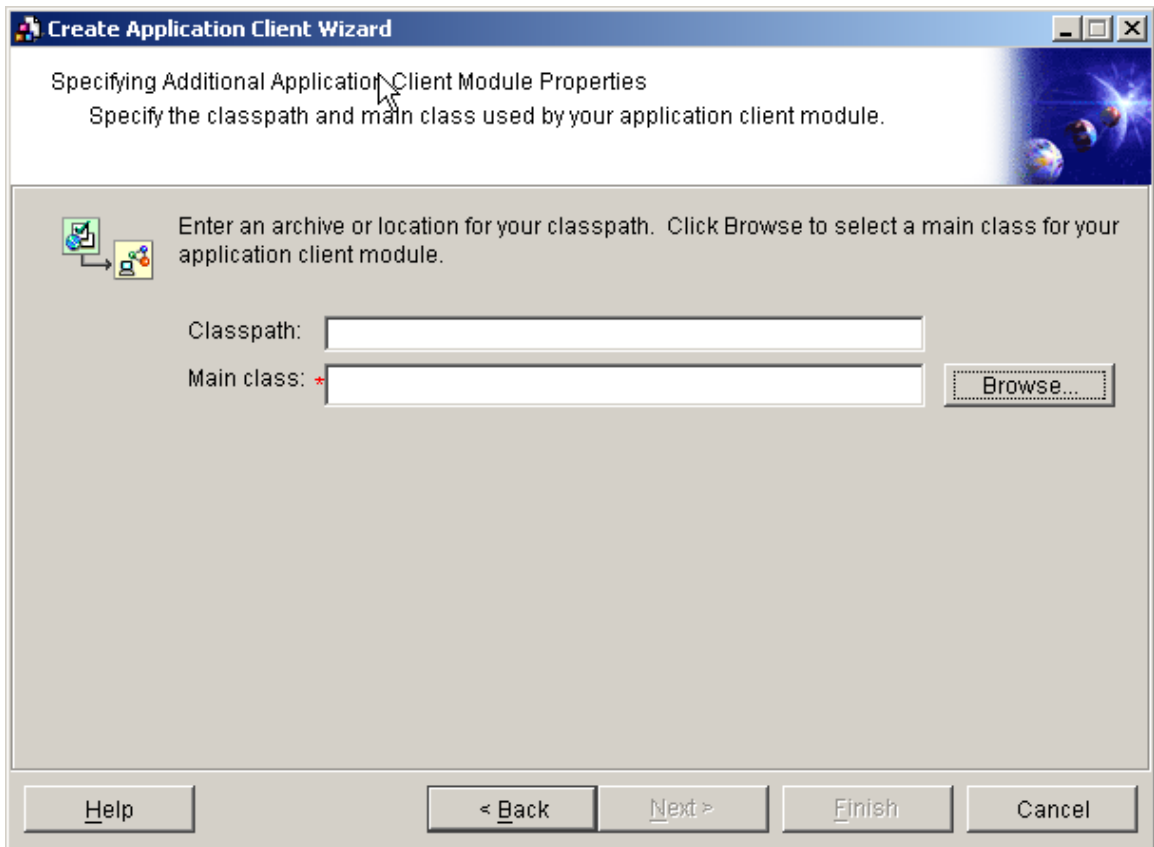
Figure 25: Adding *Client_1*

Client_1.class is the only file you are adding at this point, so press **OK** to dismiss this dialog and return to the Adding Files wizard screen. Note that your new *.class* file now appears in the list:

Figure 26: Adding files – showing Client_1

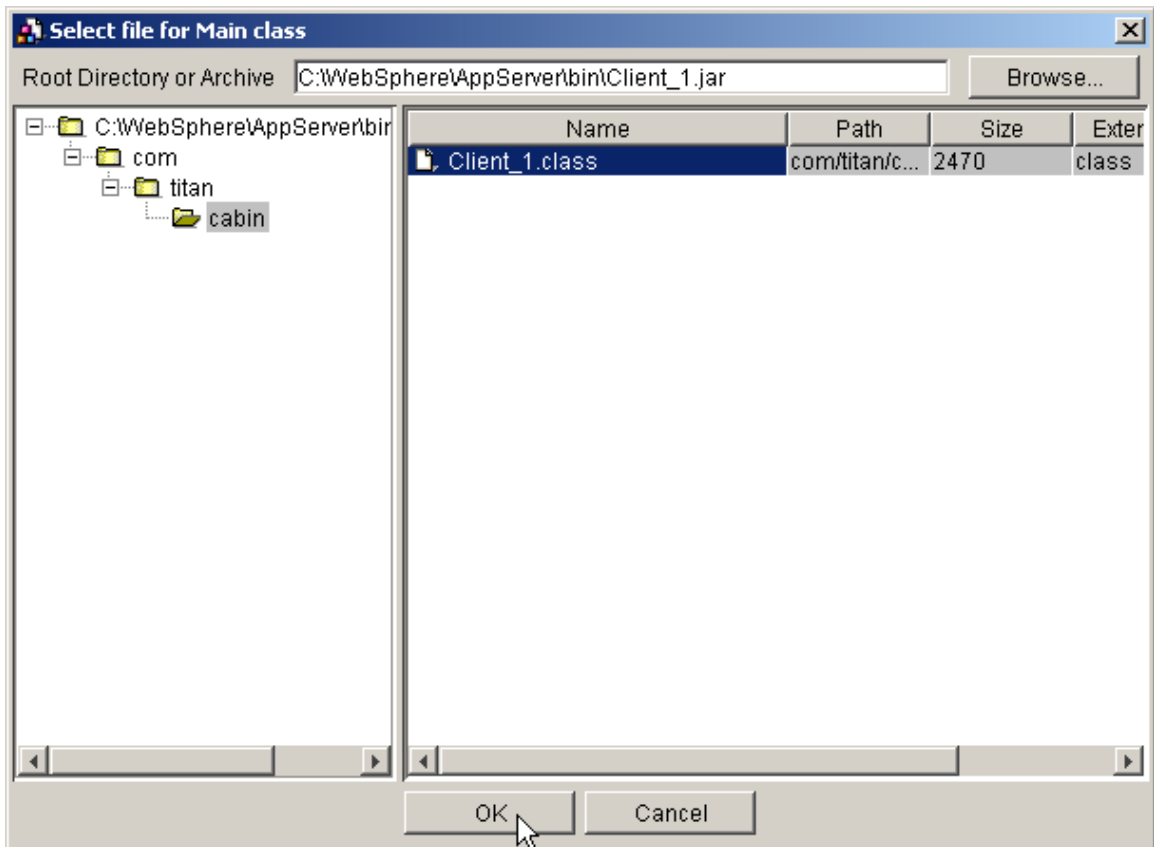
Press **Next** to indicate that you are finished with this screen. The following screen allows you to select the class whose `main()` method will be invoked to start your client application.

Figure 27: Specifying additional module properties



In this dialog you can enter a classpath the application client will use to locate any additional Java classes it needs that are not included in the application client *.jar* file. Your test program does not use additional files, so leave this field blank. You do, however, need to specify the class responsible for starting your application. Press the **Browse...** button next to the **Main class** entry field to bring up the following dialog:

Figure 28: Selecting the main class

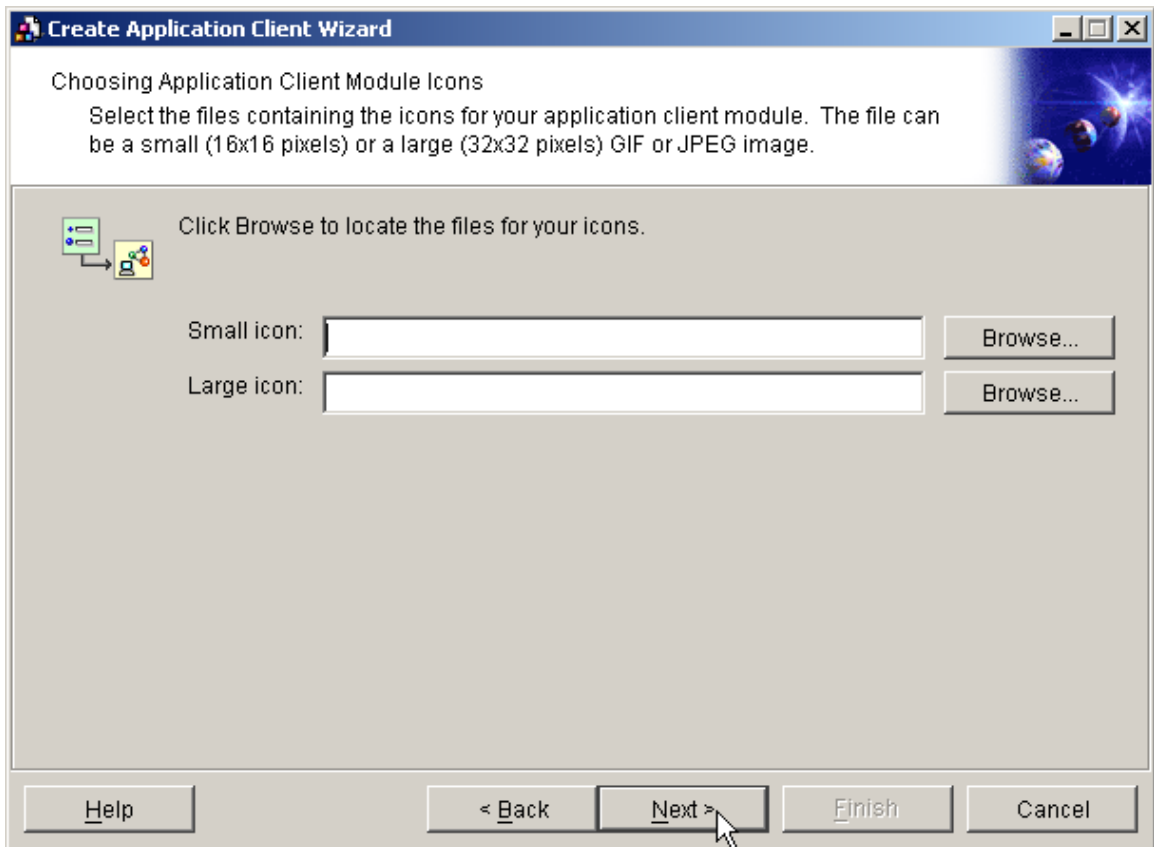


This dialog lets you navigate through the set of files that you added when building the client application to find the `.class` file of the class containing the `main` method.

- Be certain that you actually select the appropriate class – when you expand a subdirectory the topmost class will be highlighted, but it will **not** be selected. You must explicitly click on the filename to select it.

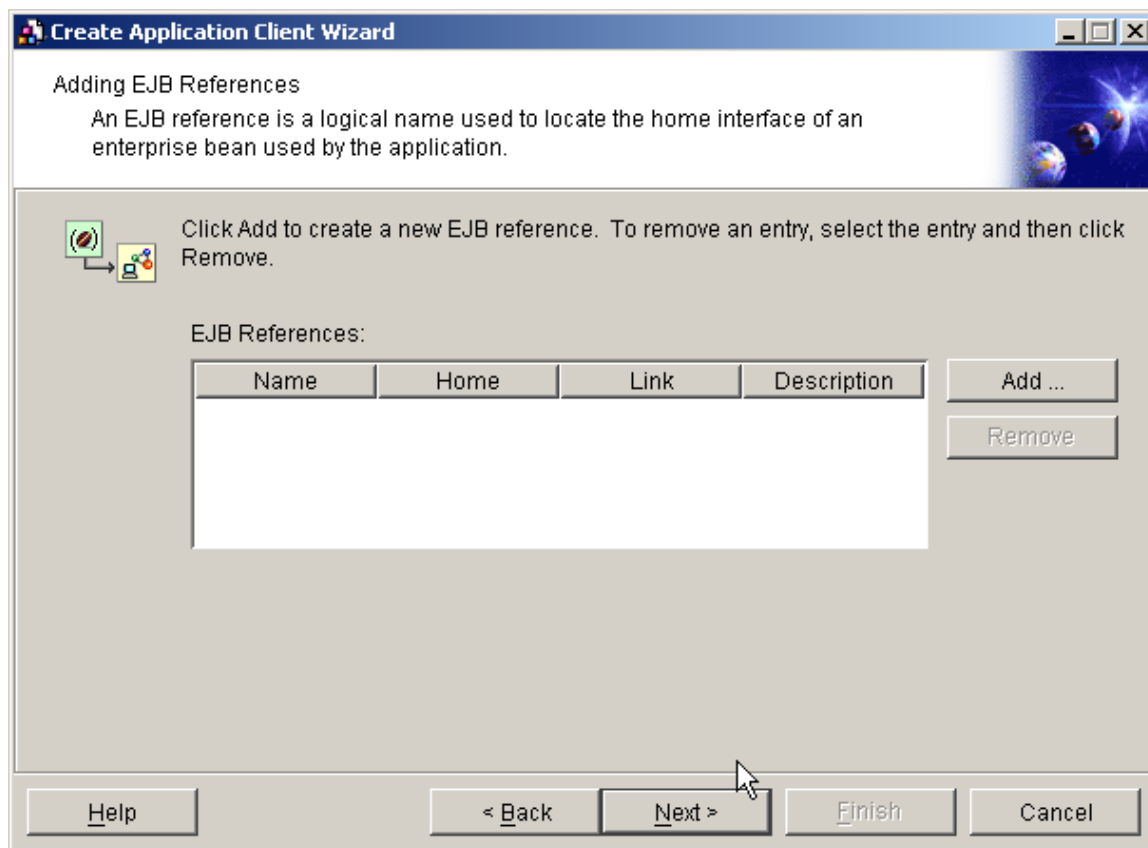
Once you have selected the appropriate class (`com/titan/cabin/Client_1.class` in this example) press the **OK** button to dismiss this dialog and return you to the Additional Properties wizard screen (Figure 27). Note that the **Main class** entry field now contains the class name you just selected, then press **Next** to move to the next page in the wizard:

Figure 29: Choosing application icons



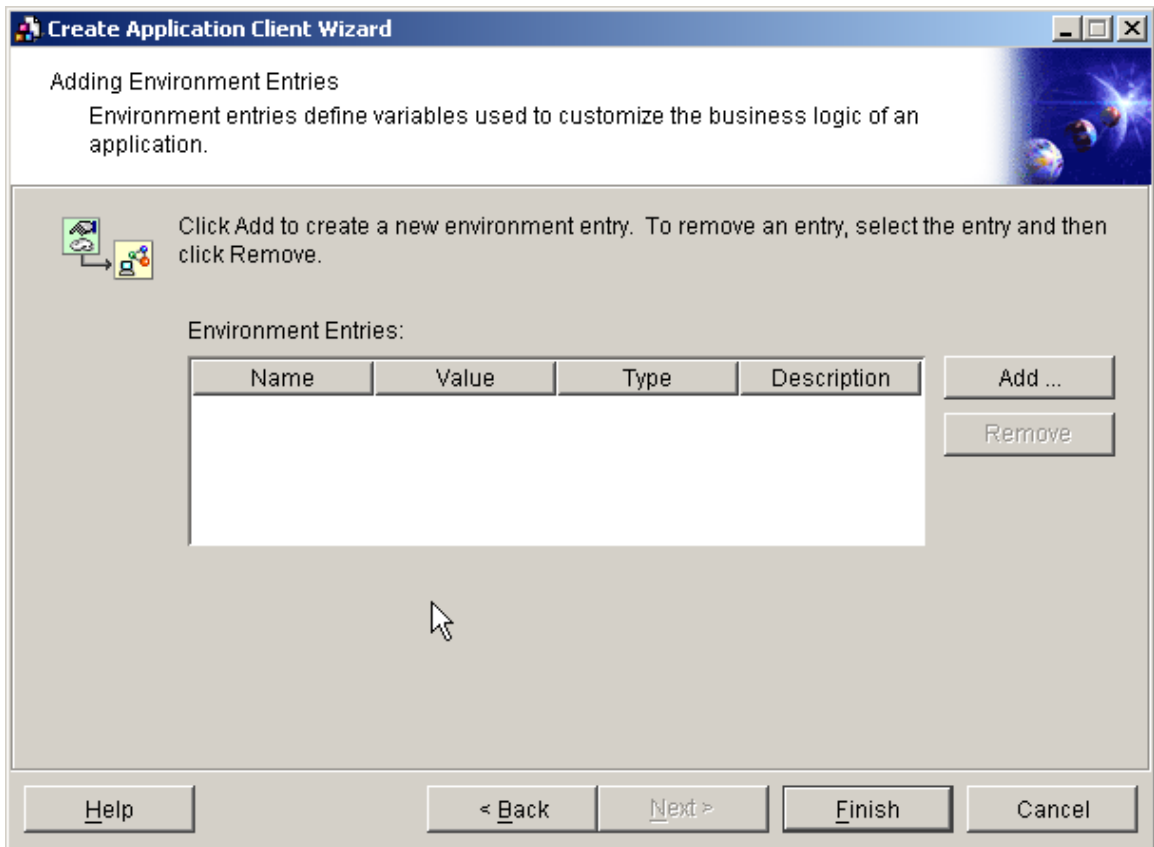
This page allows you to define icons for the application client. If you do not choose an icon, a default will be used. You are not supplying any new icons, so just press **N**ext to move to the next screen:

Figure 30: Adding EJB references



Chapter 4 of the EJB book describes how you can define in a deployment descriptor an *EJB reference*: a local “shorthand” JNDI reference to an EJB Home. Your code can then refer to an EJB by this name regardless of the actual JNDI name by which it is known in the deployed context.

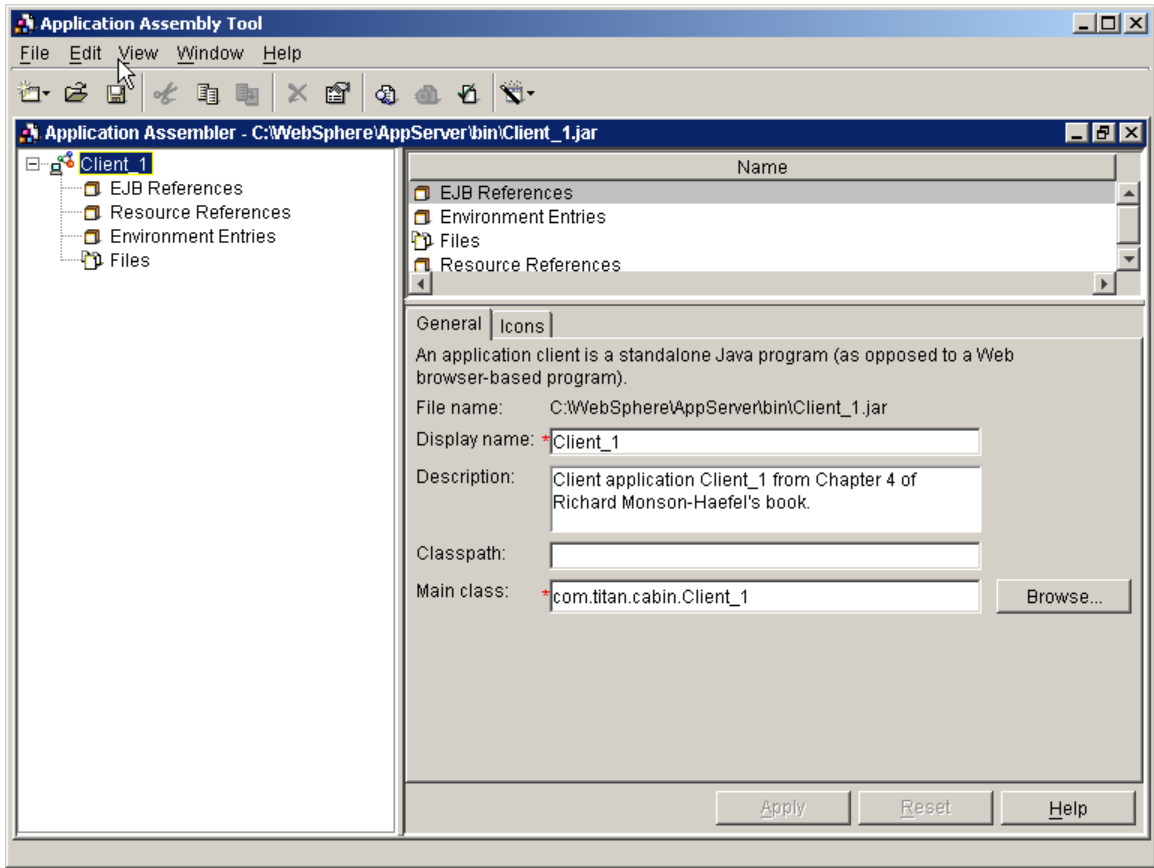
This feature can simplify your life. For example, you can switch from a “test” EJB server to a “production” EJB server by changing the mapped EJB at deployment time, without having to change your code. The session EJB you will deploy in Exercise 4.2 takes advantage of this facility, but the code for Exercise 4.1 does not, so just press **Next** to move on. The next screen allows you to add *resource references* to an application client *.jar* file. Like an EJB reference, a resource reference is a shorthand name; but the reference is to a resource like a data source or a JMS queue. You will see how to use resource references in the chapter on bean-managed persistence. In this case, you do not have any, so can press **Next** to move on to the next screen:

Figure 31: Adding environment entries

In many applications you will need to specify *environment entries*, fixed pieces of information that the application will use, such as a directory name, or a help file location. You are not specifying any environment entries, so you will not add any to this list.

At this point you are finished with this wizard. Press the **Finish** button to move back to the main screen of the AAT, and examine the new application client you just added:

Figure 32: AAT showing completed client



You are now nearly finished using the AAT to construct your first client *.jar* file. Only one more thing must be done. Note that the filename shown above indicates that your *.jar* file has been saved in the *WebSphere\AppServer\bin* directory. This is the default location for these files when created by the AAT wizards.

- It is a best practice to save these files in the directory where you earlier created the EJB *.jar* file; e.g., *<InstallDrive>\EJBWorkbook\Source\Chapter4\dev*.

Select **File→Save As...** and save the new *Client_1.jar* file in the recommended directory. You will need to find it here when you perform the next step, which is to assemble the EJB *.jar* file and your new client *.jar* file into an enterprise archive File.

Now that your first client application is constructed and saved properly, you need to repeat all these tasks for the *Client_2* application. To invoke the application client wizard a second time, select **File→Wizards→Create Application Client Wizard**. Follow the same steps again,

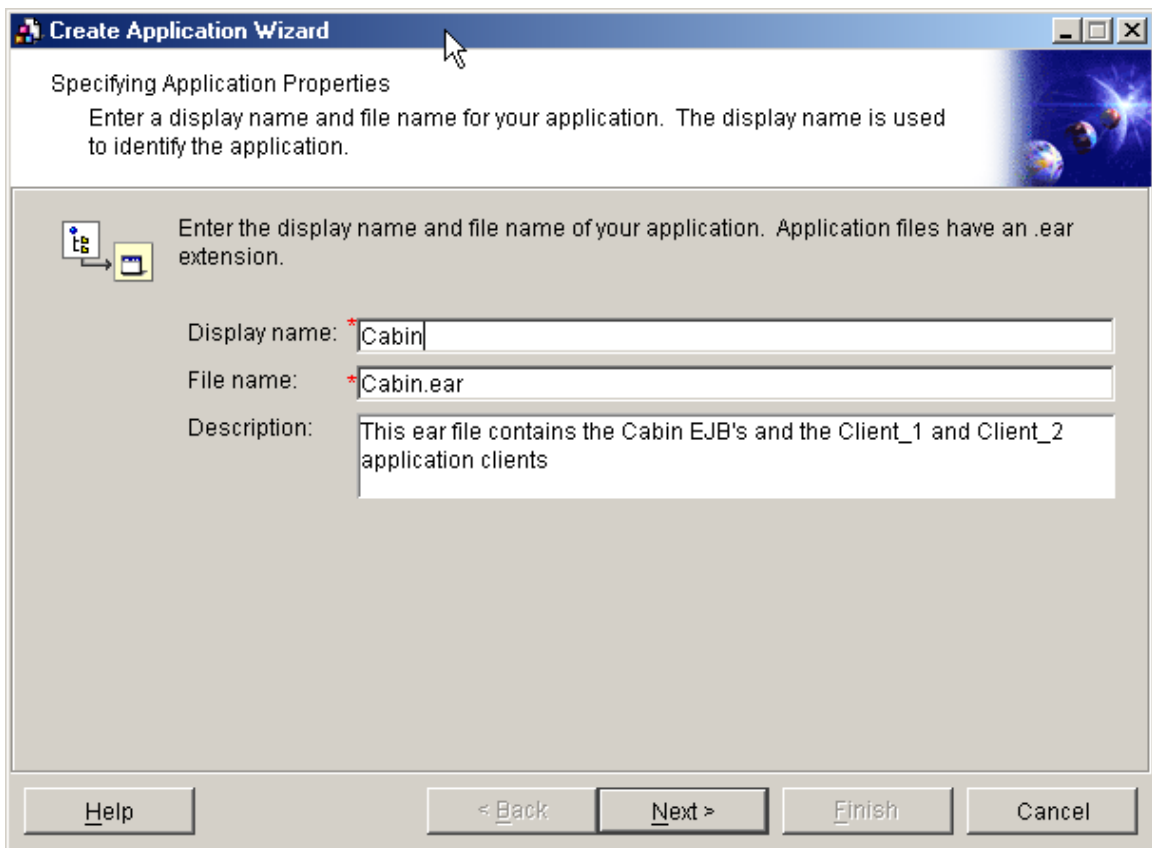
simply substituting *Client_2* wherever you used *Client_1* (in the *.jar* file name, the display name, the name of the *.class* file, etc.)

Step 4: Use the AAT to assemble an enterprise archive (.ear) file

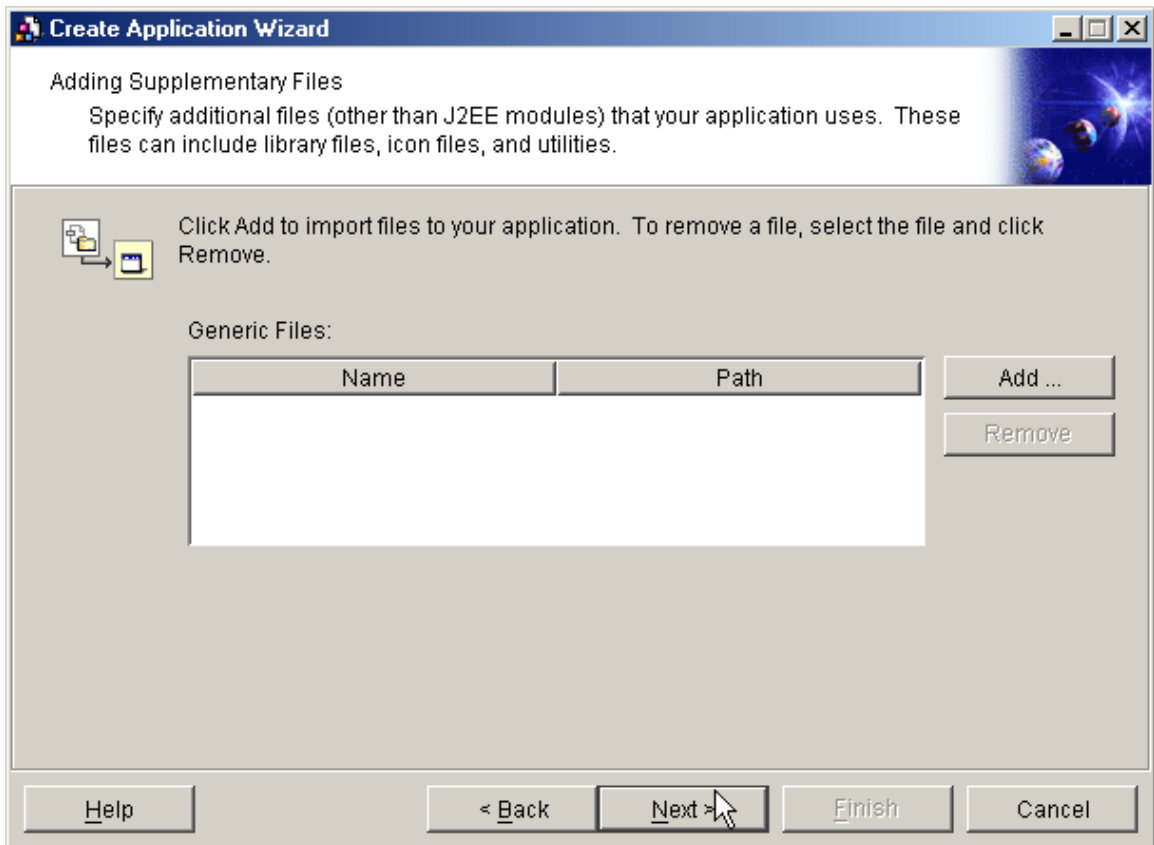
The following step will complete this section's tour of the features of the AAT. Now that you've seen how to assemble EJB *.jar* files and application client *.jar* files, you are ready to see how the two are assembled into an enterprise archive file.

Begin by invoking the **Create Application Wizard** either from the **File→Wizards** menu or from the AAT's startup page. The first page of this wizard is shown below:

Figure 33: Specifying application properties

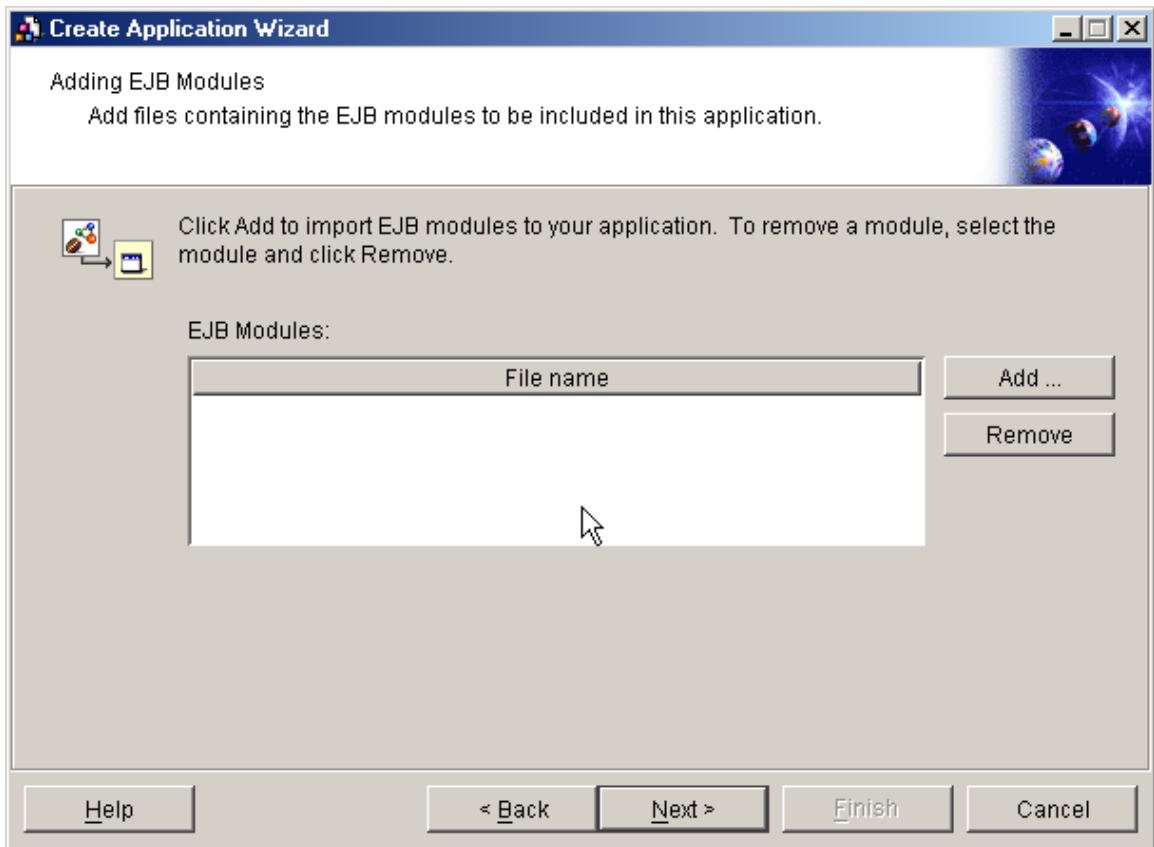


In this wizard set the **Display name** to be **Cabin** and the **File name** to be **Cabin.ear**, as shown in the dialog. Enter any text you like as the description. Then Press **Next** to move to the next wizard page:

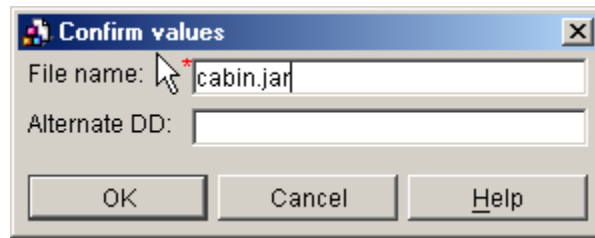
Figure 34: Adding supplementary files

This wizard allows you to add any additional files to your *.ear* file. You are not using any files other than J2EE modules, so press **Next** to move on to the next page. There you have a chance to select icons for your application. As before, you have no need to specify special icons for your J2EE modules, so press **Next** to move on to the following page:

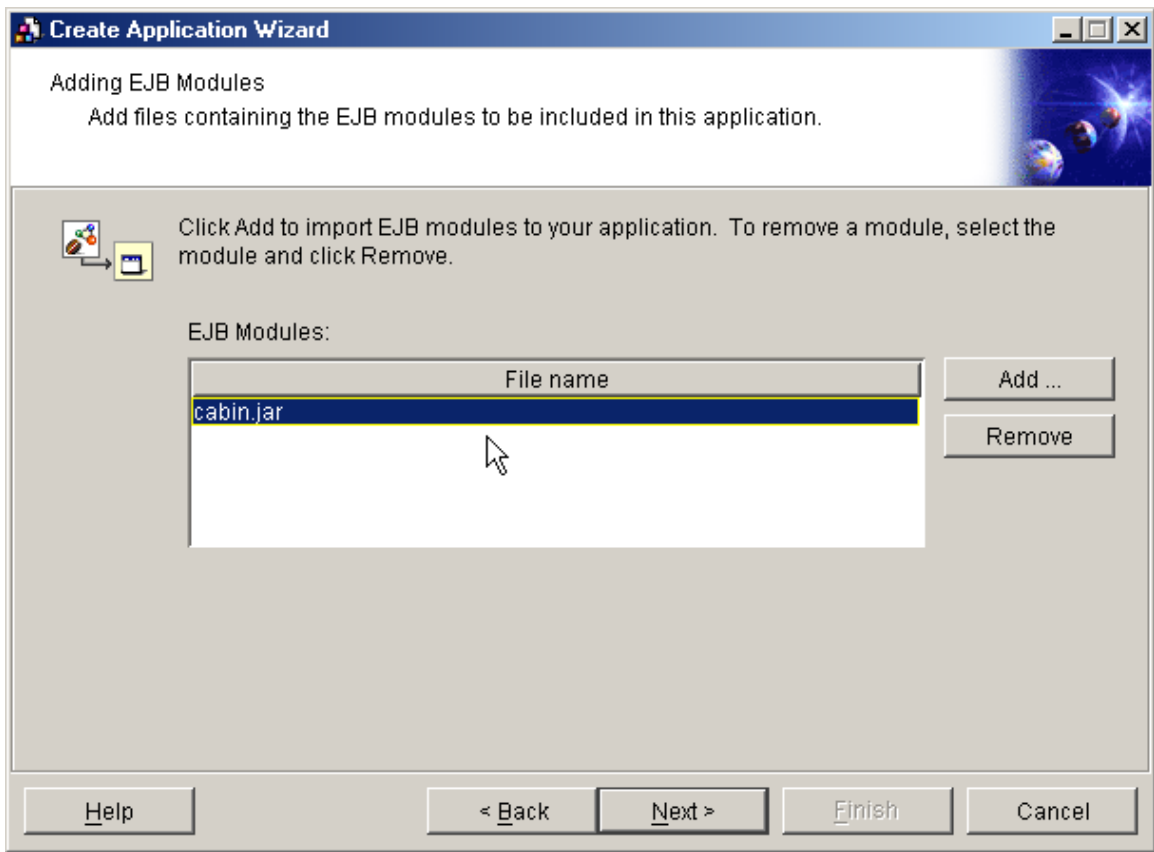
Figure 35: Adding EJB modules



Finally you are ready to add some components to your application. This wizard page is very similar to ones you have seen for adding files to EJB *.jar* files and client *.jar* files. As in those steps, begin by pressing the **Add...** button. In the standard file-selection dialog that appears, browse to the directory that contains the *cabin.jar* file created earlier and select the file. After selecting the file, press **Open**, and the following dialog will appear:

Figure 36: Confirming values

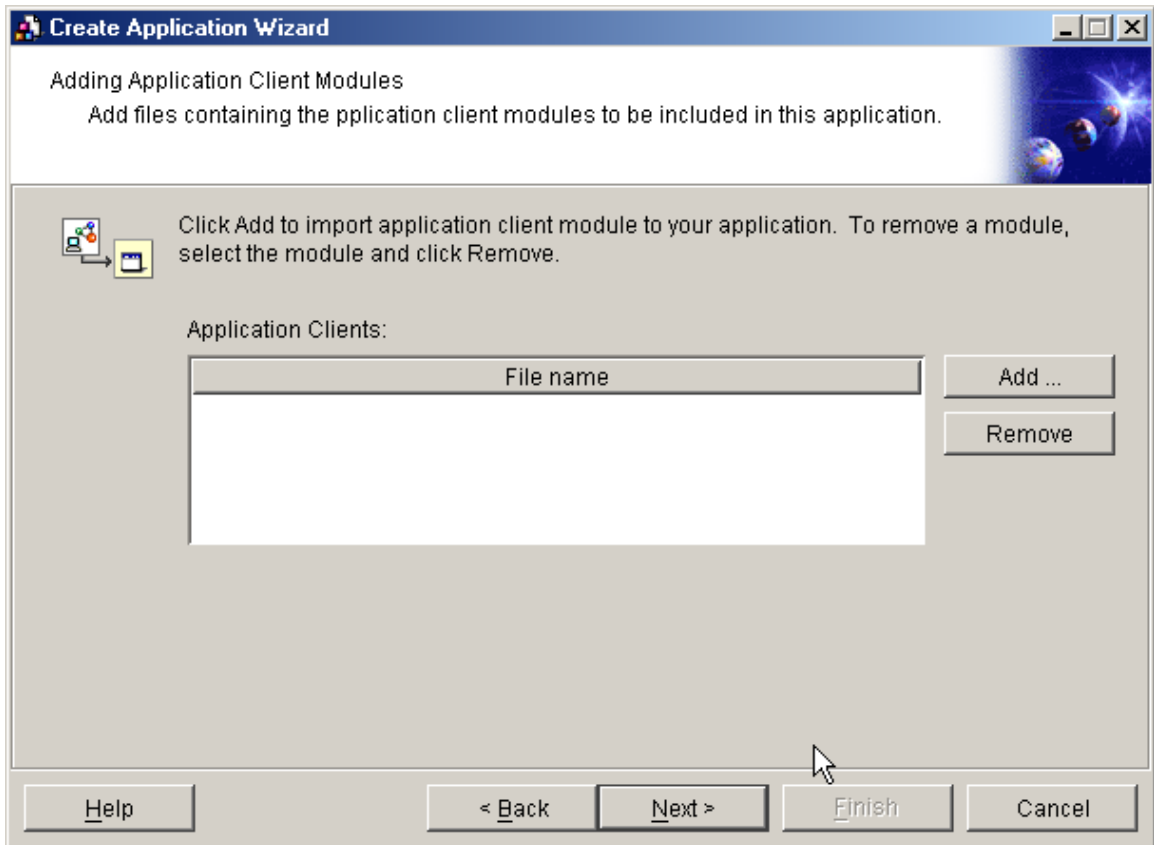
This dialog allows you to select an alternate deployment descriptor for the EJBs in this EJB *jar*, if one is available. In this case, you want to accept the default, so press **OK** to dismiss the dialog. At this point you will return to the **Adding EJB Modules** page, where you will see the *cabin.jar* file selected:

Figure 37: Adding EJB modules – Cabin EJB jar selected

In this case, you want to deploy only a single EJB *.jar* file, so press **Next** to move on to the next page, which allows you to add web archive files.

You do not have any web modules to add to your application, so press **Next** to move on to the next page:

Figure 38: Adding application clients



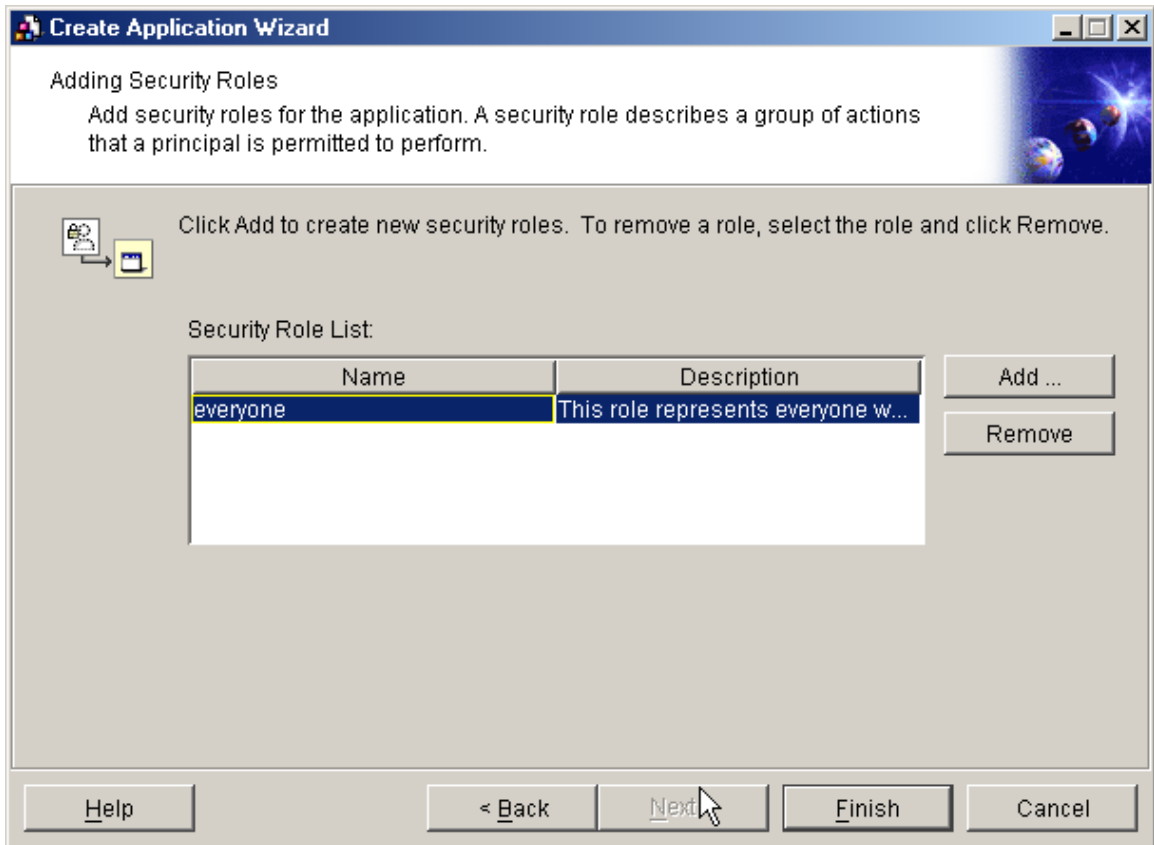
This wizard page (again, much like others you have seen) allows you to add application client *.jar* files to your *.ear* file. You will use this wizard page to add your two client modules to the *.ear* file you are creating, following the same procedure you did a moment ago to add *cabin.jar*:

1. Press the **Add...** button to bring up the file-selection dialog
2. Select *Client_1.jar* and press **Open**
3. In the **Confirm Values** dialog, press **OK** to accept the default deployment descriptor.

The first client *.jar* has now been added to the *.ear* file. Repeat the last three steps to add *Client_2.jar*. When you are finished, both should appear in the Adding Application Client Modules page.

Now that you have added both client modules, press the **Next** button to move to this wizard page:

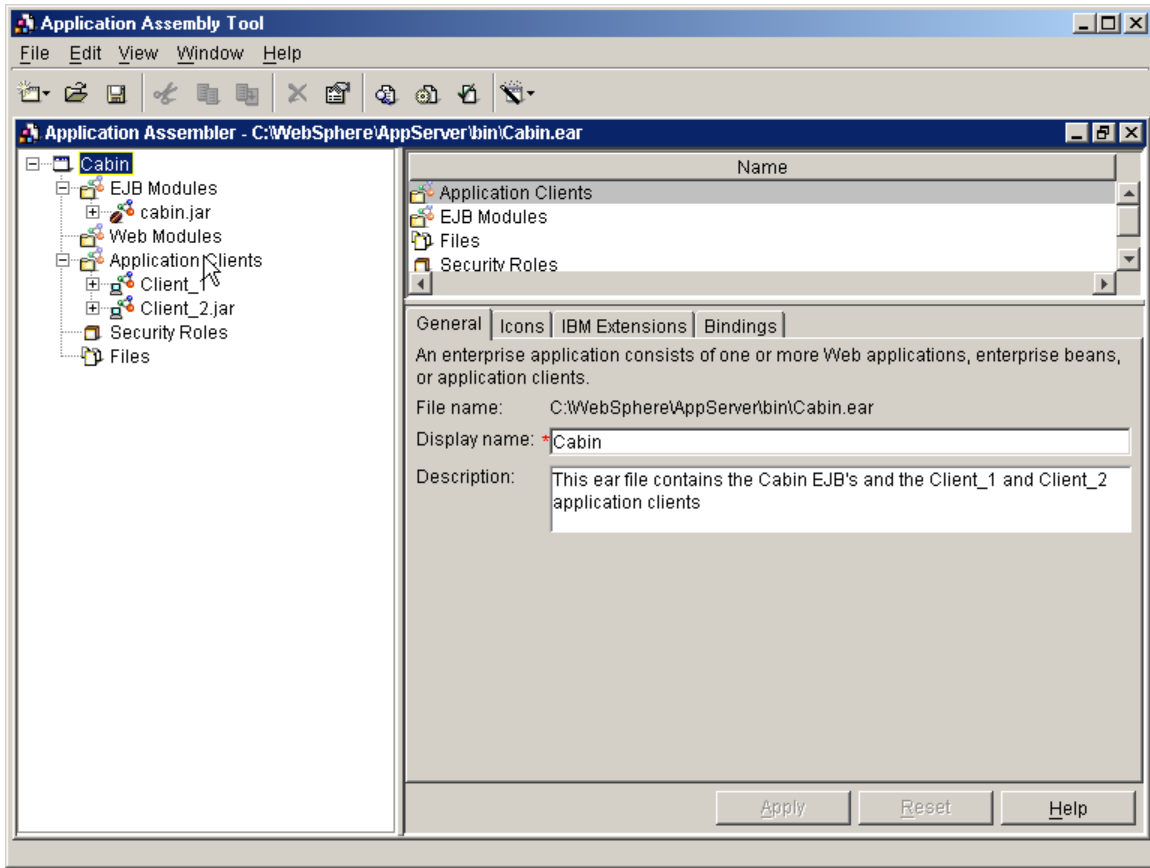
Figure 39: Adding security roles



This page allows you to add security roles to your application or remove them from it. You can see the security roles present in the EJB *.jar* files on this page. You simply want to take the defaults, so press the **Finish** button to complete building your *.ear* file.

At this point, you should see the AAT main view, displaying your new EJB *.jar* file and application client files thus:

Figure 40: AAT with EJB and client files

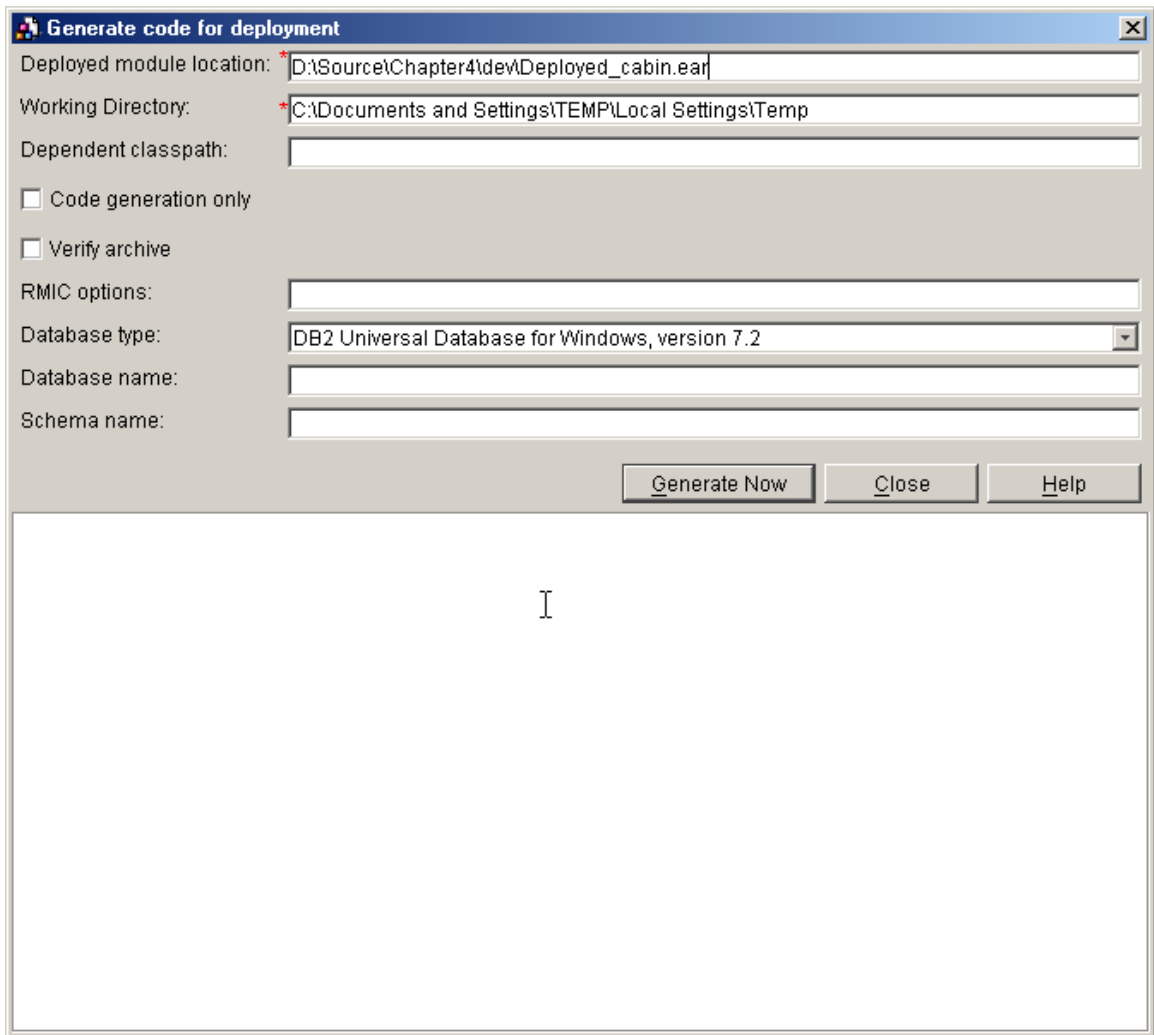


You are now finished creating your first `.ear` file. Select **File→Save As...** and save the file to your working directory under the name `cabin.ear`.

Step 5: Use the AAT to generate deployment code for the .ear file

Before you leave the AAT, there is one more task to accomplish. So far you have created a J2EE application archive file that contains the compiled code and deployment descriptors for your Cabin EJB and the two client programs; but you have not generated any of the deployment code that the client programs will need to connect to the EJB running on the WebSphere AEs application server.

First, make sure that you still have your new `cabin.ear` file selected in the AAT. If you have closed the AAT, then use the **Existing** tab of the startup page to select the file. From the **File** menu select **Generate Code for Deployment**. The following window will appear:

Figure 41: Generating code for deployment – start

This window deserves a bit of explanation, because it can be used to illustrate a number of concepts about how WebSphere AEs works. There are a number of text fields and checkboxes on this window. In general, the top several fields are concerned with the generation of the RMI-IIOP stubs and skeletons for the EJBs in your *.ear* files. You will use the bottom three text fields only if you are generating CMP EJBs, to provide information to the WebSphere AEs code generator that it needs to generate the persistence code for those CMP EJBs. The Help file for this screen provides a detailed explanation of the meaning of each of these fields and checkboxes, but here is a succinct explanation of each:

- ◆ **Deployed module location** – the location where the deployed *.ear* file (the one containing the generated code) will be placed. The default is to place the deployed *.ear* file in the same directory as its source, with *Deployed_* prepended to the file name.
- ◆ **Working directory** – The working directory for temporary file storage
- ◆ **Dependent classpath** – A classpath that will be used in the generation of the *.ear* file. Dependent classes are ones that are referenced by the EJBs in your *.ear* file, but not contained in that file. An example might be a common *.jar* file that is used by multiple EJBs deployed into several *.ear* files.
- ◆ **Code generation only** – If this box is checked, only code generation will be performed; e.g., the RMI Compiler will not be invoked.
- ◆ **Verify archive** – If this is checked, the AAT will verify the EJB archive before generating the deployment code.
- ◆ **RMIC options** – This field can contain options used by the *RMI Compiler (RMIC)*. For information on available options, either read the Help file or type `rmic` at the command line.
- ◆ **Database type** – This drop-down box allows you to select from a list of databases that WebSphere AEs supports for its CMP EJBs. If your database is not on this list, you may select either “Generic SQL/92” or “Generic SQL/99”. IBM does not guarantee, however, that CMPs will work with unsupported databases.
 - If you are going to use a database other than DB2 for these examples, be sure to select the correct database from this list.
- ◆ **Database name** – This is the name of the database that will be used when generating the SQL persistence code, and when specifying the generated DDL file you will use in the following step.
- ◆ **Schema name** – This is the name of the database schema (e.g., `EJB.Tablename`) that will be used in the generated SQL persistence code and the generated DDL file.

In this simple case – assuming you are using DB2 as your database – you can accept all of the defaults for this window. Simply press the **Generate Now** button, and the code generator will do the rest. You should see a large amount of information displayed in the console on the lower portion of the screen, several portions of which deserve explanation. Near the top of the text you will see the following lines:

```
Creating Top-Down Map...
  Initializing...
  Mapping "CabinEJB"...
  Saving...
Generating deployment code
  Processing CMP entity bean: CabinEJB
```

WebSphere AEs is indicating that it is using its default approach to mapping this CMP EJB to a database table. WebSphere AEs supports three methods for mapping CMP EJBs to a database. They are:

- ◆ *Top-down* – The information in the EJB is used to create a database table that corresponds to the managed fields of the CMP EJB.
- ◆ *Meet-in-the-middle* – There is a pre-specified correspondence between the managed fields in the CMP EJB and the columns in one or more database tables.
- ◆ *Bottom-up* – EJB fields are created for the columns in a database table.

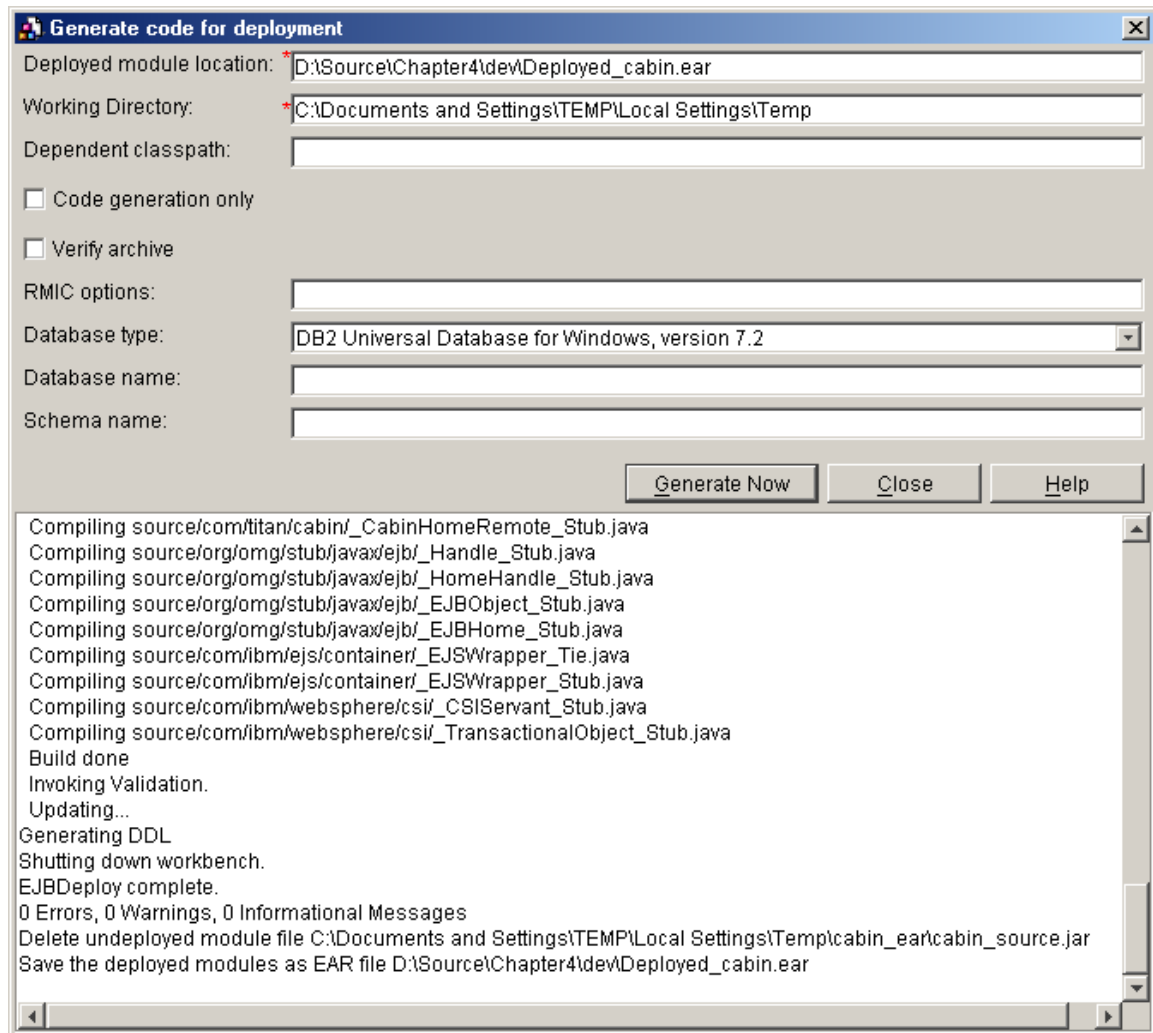
The default is top-down. If you want to use a mapping other than top-down then you will need to follow carefully the instructions found in the product documentation for building the XML files necessary to perform other types of mapping. Building these mapping files is a complex chore, so you will want to use the other two mapping types only inside a tool like VisualAge for Java or WebSphere Studio Application Developer. In this workbook, we will always use top-down mapping.

Following the declaration of the CMP mapping information, there are several sections that indicate that the AAT is generating the *.java* files that correspond to the CMP persistence engine, and the RMI-IIOP code necessary to make this EJB available over the network. There will then be lines that indicate that the RMIC compiler has been invoked, and several other sections indicating the progress of the code generation and compilation. The final section of the text should contain the following lines:

```
Shutting down workbench.  
EJBDeploy complete.  
0 Errors, 0 Warnings, 0 Informational Messages  
Delete undeployed module file C:\Documents and Settings\TEMP\Local  
Settings\Temp\cabin_ear\cabin_source.jar  
Save the deployed modules as EAR file  
D:\Source\Chapter4\dev\Deployed_cabin.ear
```

If you see any other messages (e.g., errors, warnings, or informational messages), resolve them before proceeding. Many errors encountered in this stage are due simply to not including the right files in the constituent *.jar* files of your EJB. When you are finished, your screen should look like the following:

Figure 42: Code generation successful



After code generation is completed, press the **Close** button to dismiss this dialog, then exit the AAT.

Step 6: Use the DB2 Command Line Processor to create the Cabin bean table

You are now ready to take another step described in Chapter 4 of the EJB book: creating the **CABIN** table in the database. We will, however, diverge from the EJB book here in a subtle way,

due to the way in which WebSphere AEs operates. Remember from the previous step that the code generation for CMP beans in WebSphere AEs allows three types of mapping to connect a CMP bean to a relational database table. Because we chose the default top-down mapping, the AAT has already automatically generated the appropriate SQL DDL to create the database table for the database. WebSphere does this because databases differ in the way in which their `CREATE SQL` operates – for instance, the supported SQL datatypes for CMP mapping are not exactly the same under DB2 on OS/390 as under Oracle. Because WebSphere automatically generates the DDL for each target database, you can be assured that the CMP mappings will be correct if you use the generated DDL. If you don't use the generated DDL, WebSphere may not be able to correctly read or write the columns from the database properly.

You can find the generated DDL inside the *cabin.jar* file that is inside the deployed *.ear* file you just created. During the top-down mapping process the AAT code generator places a file named *table.ddl* inside this file along with the other compiled code for the persistence classes and RMI-IIOP classes. To examine this DDL, unzip the *cabin.ear* file to a temporary directory (using an unzip utility or `jar -xvf`), and then unzip the deployed *cabin.jar* file just extracted from the *.ear* file. The *table.ddl* file will be in the *META-INF* directory.

The content of this file (when generated for DB2 7.2) is shown below:

```
CREATE TABLE CABINEJB
  (ID INTEGER NOT NULL,
   NAME VARCHAR(250),
   DECKLEVEL INTEGER,
   SHIPID, INTEGER,
   BEDCOUNT INTEGER)

ALTER TABLE CABINEJB
  ADD CONSTRAINT CABINEJBPK PRIMARY KEY (ID)
```

If you chose a different database in the previous step, you may see different DDL in this file. You will have to use the tools of your chosen database to create the tables for this example (and all other examples) from the DDL found in this file.

If you are using DB2, you now need to use the DB2 Command Line Processor (CLP) to execute these SQL statements. Begin by opening the CLP. From the Windows **Start** menu, select **Programs→IBM DB2→Command Line Processor**.

Then issue the following command:

```
CONNECT TO SAMPLE
```

After successfully connecting to the `SAMPLE` database, you can execute the `CREATE` statement shown above (it is easiest just to type in the whole statement on one line). Then execute the `ALTER TABLE` statement above. When you have finished executing these three statements, you may either close the DB2 CLP, or leave it open to verify the correct execution of some of the following steps

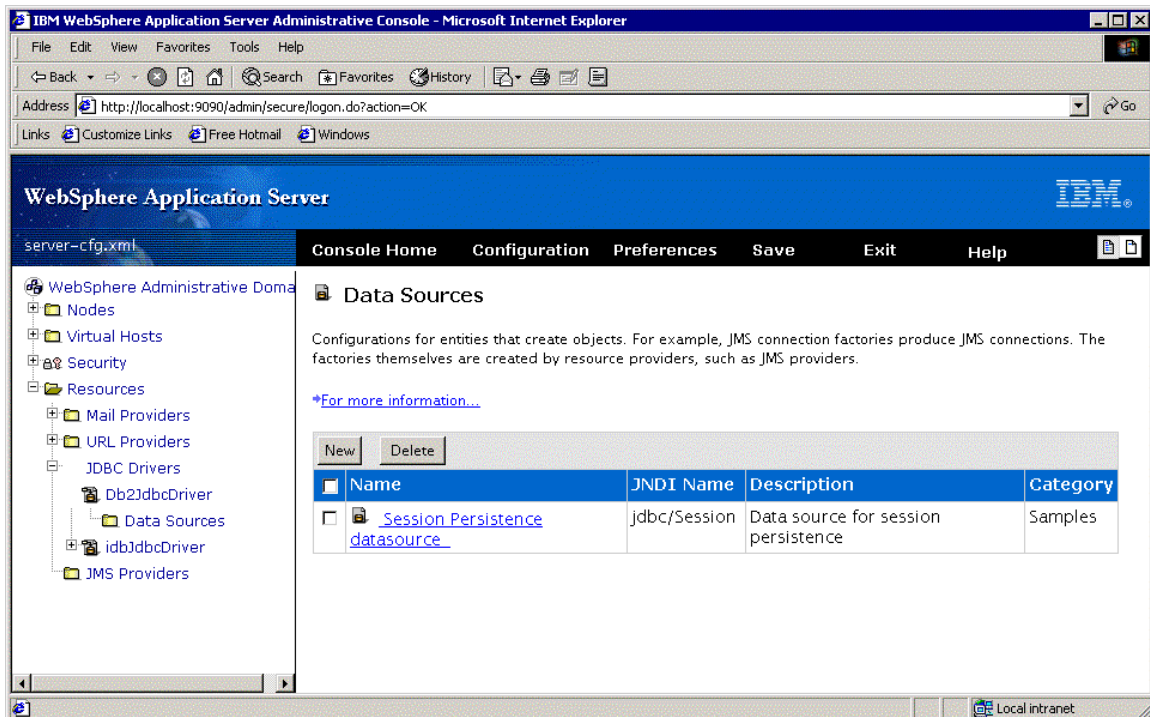
- As an alternative to all that typing, once you have unzipped the *table.ddl* file, you can open a DB2 Command Window and type `db2 -t -f table.ddl`

Step 7: Use the Administrative Console to create a DB2 data source

Now that you've created the table in the database, the next step is to create a data source in the WebSphere Administrative Console to connect to that data source. Remember that in JDBC 2.0 data sources are the client view of a database connection pool. They are used by the generated EJB code to connect to the database in a controlled and efficient way.

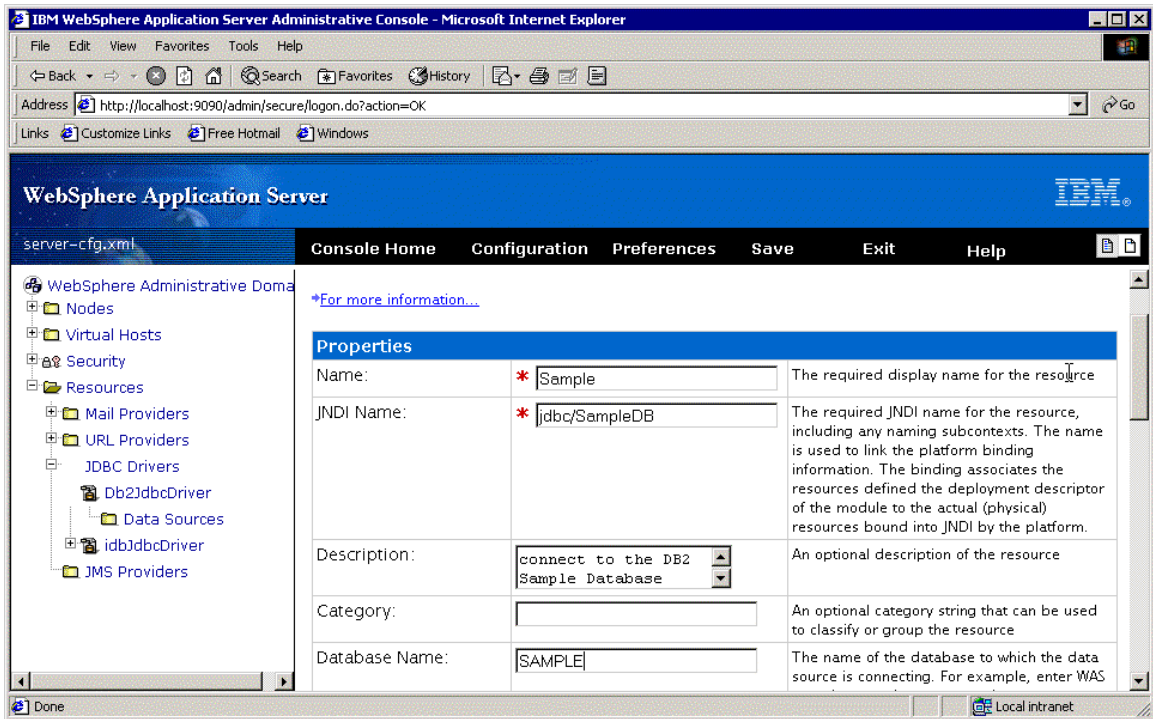
Begin by making sure that the application server is started (you can start it from the First Steps window, or from the Start menu). Then start the Administrative Console. Using the tree view on the left side of the Administrative Console, navigate down to the **Resources** tab, and then to the **JDBC Drivers** tab below that, and finally to the **Db2JdbcDriver** and the **Data Sources** tab. When you have selected the **Data Sources** tab, you will see the following page on the right side of the Administrative Console:

Figure 43: Data sources



To add a new data source to this list, press the **New** button. You will see the following page:

Figure 44: Adding the sample data source

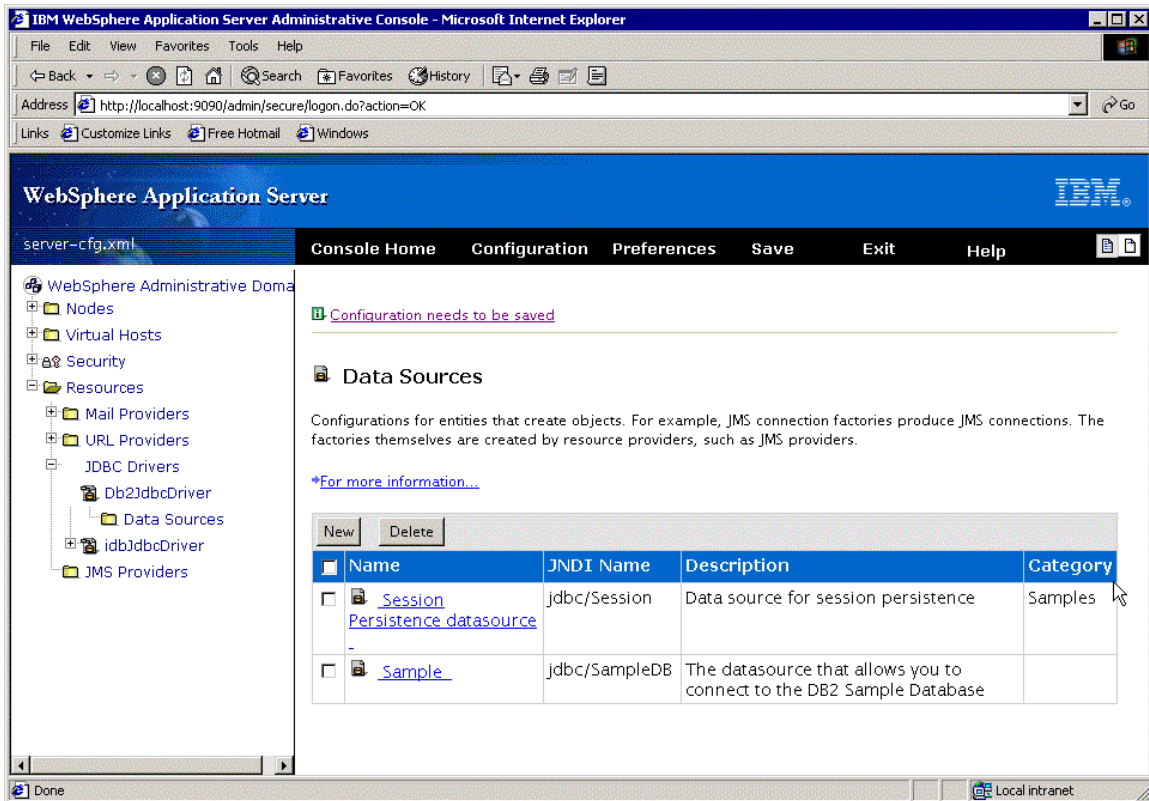


Here you will define your new data source. Fill in the following text fields and leave the rest blank (their defaults are fine):

- ◆ **Name:** This is the name used in the **Data Sources** display; enter **Sample**.
- ◆ **JNDI Name:** This is the name used to locate this data source through JNDI; enter **jdbc/SampleDB**.
- ◆ **Description:** Enter any descriptive string you like.
- ◆ **Database Name:** This is the name of the DB2 database this data source will connect to; enter **SAMPLE**.

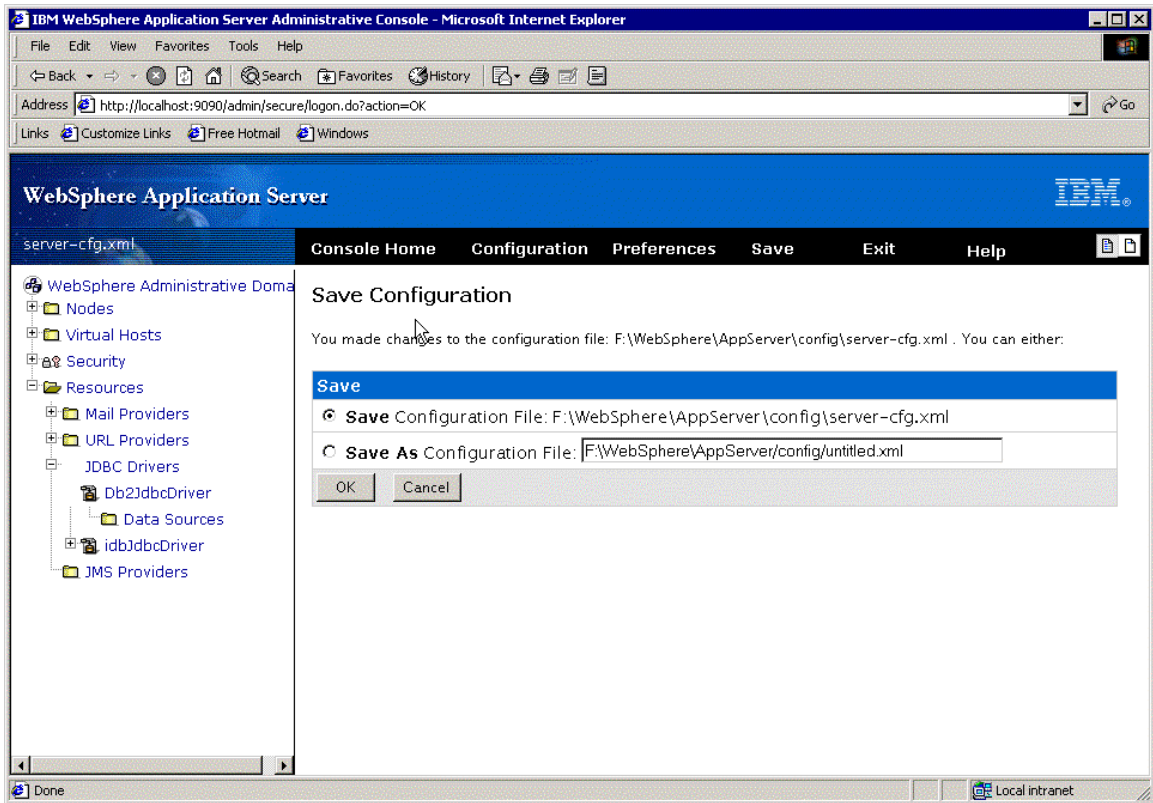
Then press **OK** to get back to the Data Sources list:

Figure 45: Data sources – to be saved



At this point the configuration objects have been changed inside the administration application, but the changes have not been saved into the XML configuration file, as that now familiar link at the top of the frame advises. Click on **Configuration needs to be saved** to go to the following screen:

Figure 46: Saving your configuration file



Simply press **OK** to save the new contents of this file.

Note that the new data source will not be created and registered in JNDI until you stop the application server and restart it, so do so now.

Step 8: Use the Administrative Console to deploy the EJB .jar file

Now that you have completed the preliminaries, you are ready to actually install your EJB into the WebSphere Application Server. Remember that in J2EE three roles that are vital in building and deploying an EJB application:

- ♦ The **Application Component Provider** is responsible for producing J2EE components such as EJBs, Servlets, and JSPs. Even though most of the work had already been done for you, you took on this role when you compiled the EJB files and assembled them into EJB .jar files.

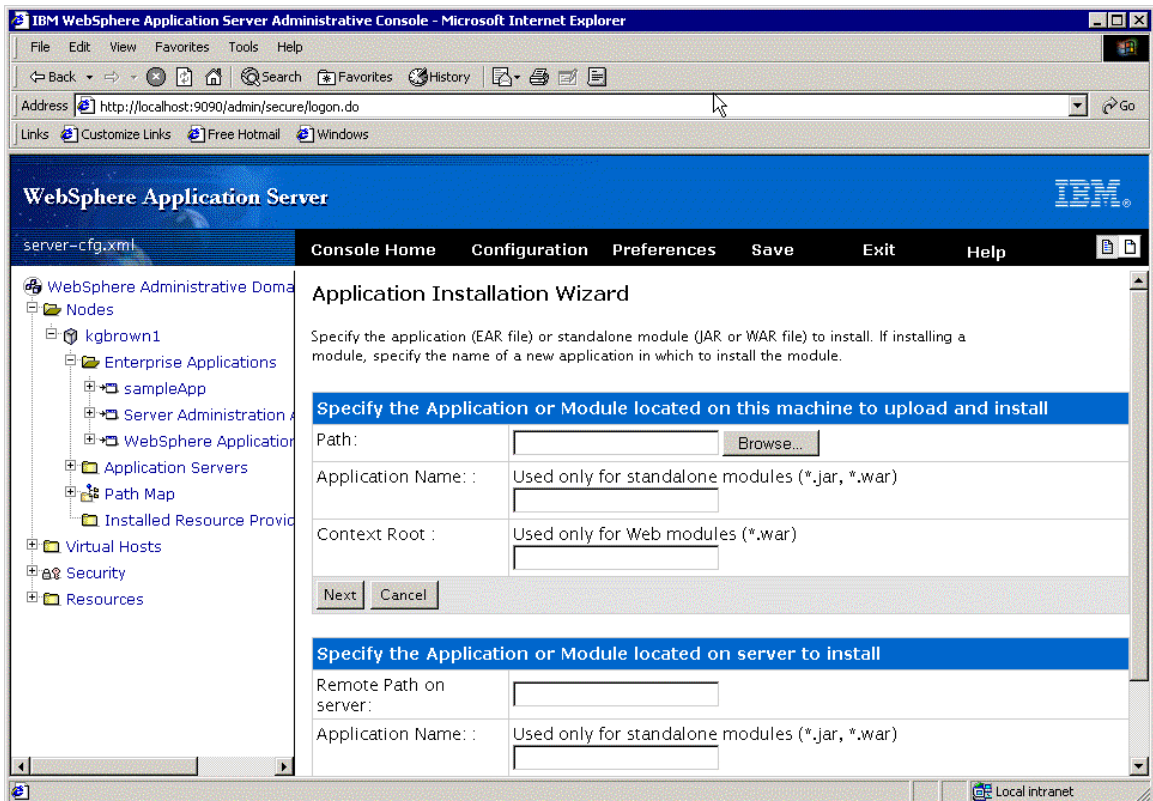
- ◆ The **Application Assembler** is responsible for assembling J2EE components into *.ear* files. You already took on part of this role when you built the deployed *.ear* file for the client side of your application, and you are taking on more of this role now.
- ◆ The **Deployer** is responsible for installing and configuring the J2EE components into the application server. These tasks include specifying the mapping between abstract security roles (defined in the *.jar* and *.ear* files) and actual people and groups, and locating and installing *.jar* and *.ear* files, among others.

You are now working mainly in the deployer role. You will carry out all of the deployer tasks in this step. Begin by ensuring that the application server process has started successfully. Then open the Administrative Console and navigate to the Application Installation wizard: open the **Nodes** tab, then the node found beneath that tab.

- ❖ The node name displayed for your machine will be its TCP/IP name.

Beneath the node, click on the **Enterprise Applications** tab, then press the **Install** button, and you will see the following wizard page:

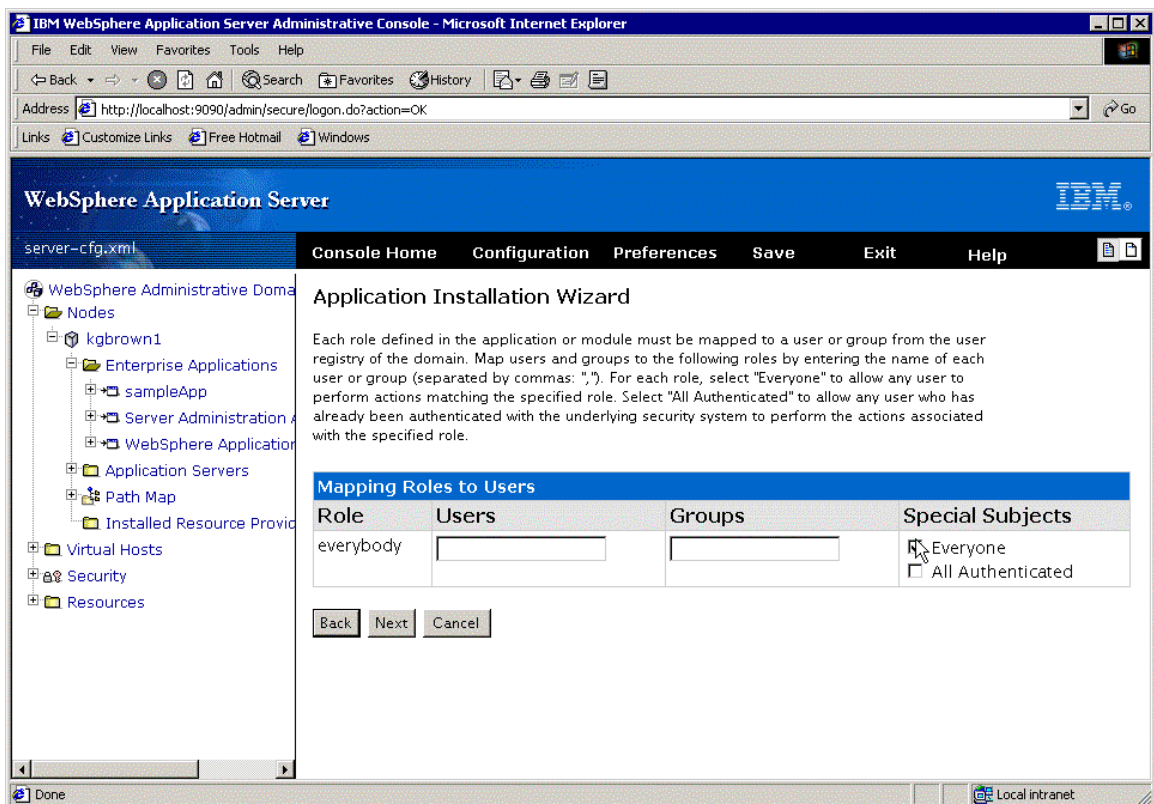
Figure 47: Application Installation Wizard



Installing an EJB *.jar* file is a simple process in WebSphere AEs. First, you must locate the EJB file to install, in one of two ways. If the file is located on the same machine the Administrative Console is running on, use the top part of the page. If, however, you are running the Console remotely and the file you wish to install is on the WebSphere server machine, use the bottom portion of the page. In this case, we are assuming that the WebSphere server and the Administrative Console are co-located, so use the top part of the wizard.

Press the **Browse...** button beside the **Path** text field to open a file browser. In the browser select the *cabin.jar* file you created earlier and press the **Open** button. In the **Application Name** field, type **Cabin** as the name of the application you will be creating. Finally, press the **Next** button to move on to the next page of the wizard:

Figure 48: Mapping roles to users and groups



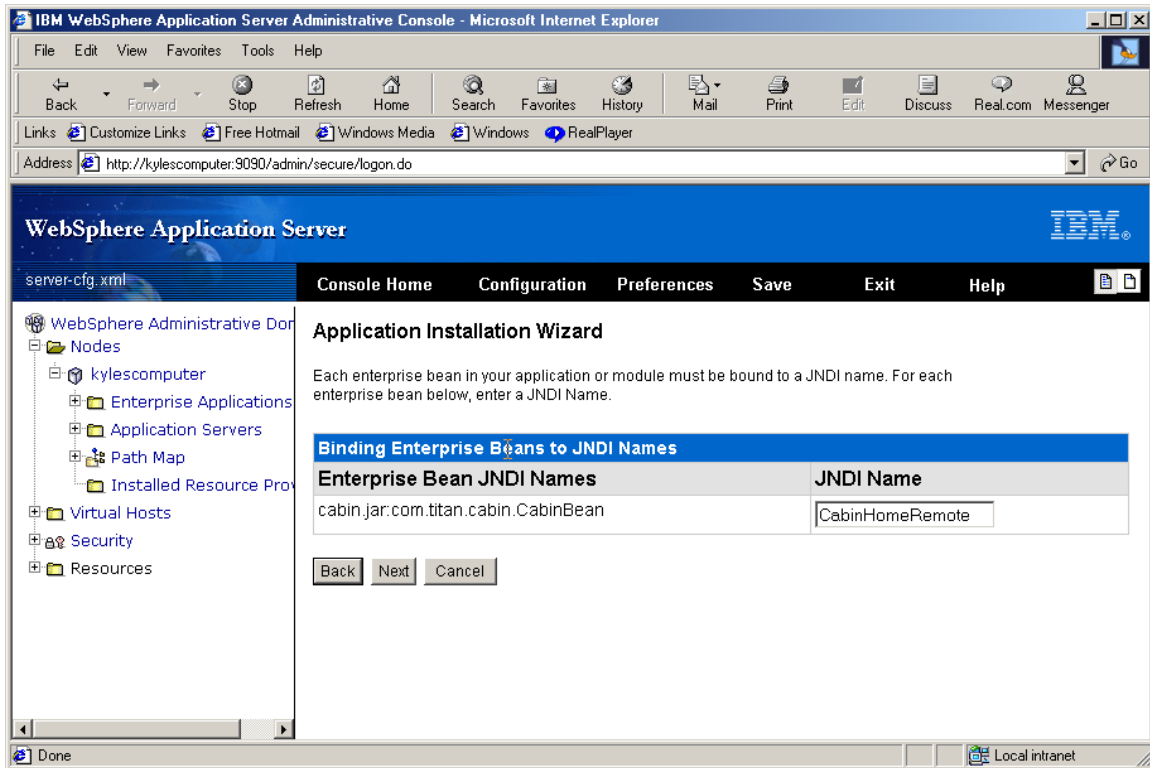
In this page you can map the roles defined in the XML configuration files (described in the EJB book) to actual users. WebSphere AEs allows you to map these roles either to users defined in the local operating system, or to a set of “special” user subjects, either “everyone” or “all authenticated users.”

In this case, map the **everybody** role to **Everyone**, to indicate that all users may perform all actions on your EJB.

- ❖ In WebSphere AEs security is globally disabled by default, so your choice doesn't matter, but when you enable security in a production server, this page becomes very important.

Press the **Next** button to move to the next wizard page:

Figure 49: Setting the JNDI name



In this page you determine the JNDI name of the EJBs in your EJB *jar* file. Because the client programs in the text expect the name of the EJB to be *CabinHome*, that is the JNDI name you will enter here, replacing the default *com/titan/cabin/CabinHomeRemote*, which is derived from the fully qualified package name of the EJB home class. Enter **CabinHome** in the text field, and press **Next** to move to the next wizard page:

Figure 50: Database settings

The screenshot shows the IBM WebSphere Application Server Administrative Console in a Microsoft Internet Explorer browser. The address bar shows the URL `http://localhost:9090/admin/secure/login.do?action=OK`. The console title is "WebSphere Application Server". The left navigation pane shows a tree structure with "Nodes" expanded, containing "kgbrown1", "Enterprise Applications", "Server Administration", "WebSphere Application Servers", "Application Servers", "Path Map", and "Installed Resource Providers". The main content area is titled "Application Installation Wizard" and contains the following sections:

Database settings

Each CMP data source reference in your application must be bound to a JNDI name. For each resource reference below, enter a JNDI Name, User ID, and Password.

Database Type:

Schema Name(Qualifier):

Mapping EJB Jar Default Data Source References to JNDI Names.

EJB Jar Name	Default Datasource JNDI Name.	User ID	Password
cabin.jar	<input type="text" value="jdbc/SampleDB"/>	<input type="text"/>	<input type="text"/>

Mapping CMP Data Source References to JNDI Names

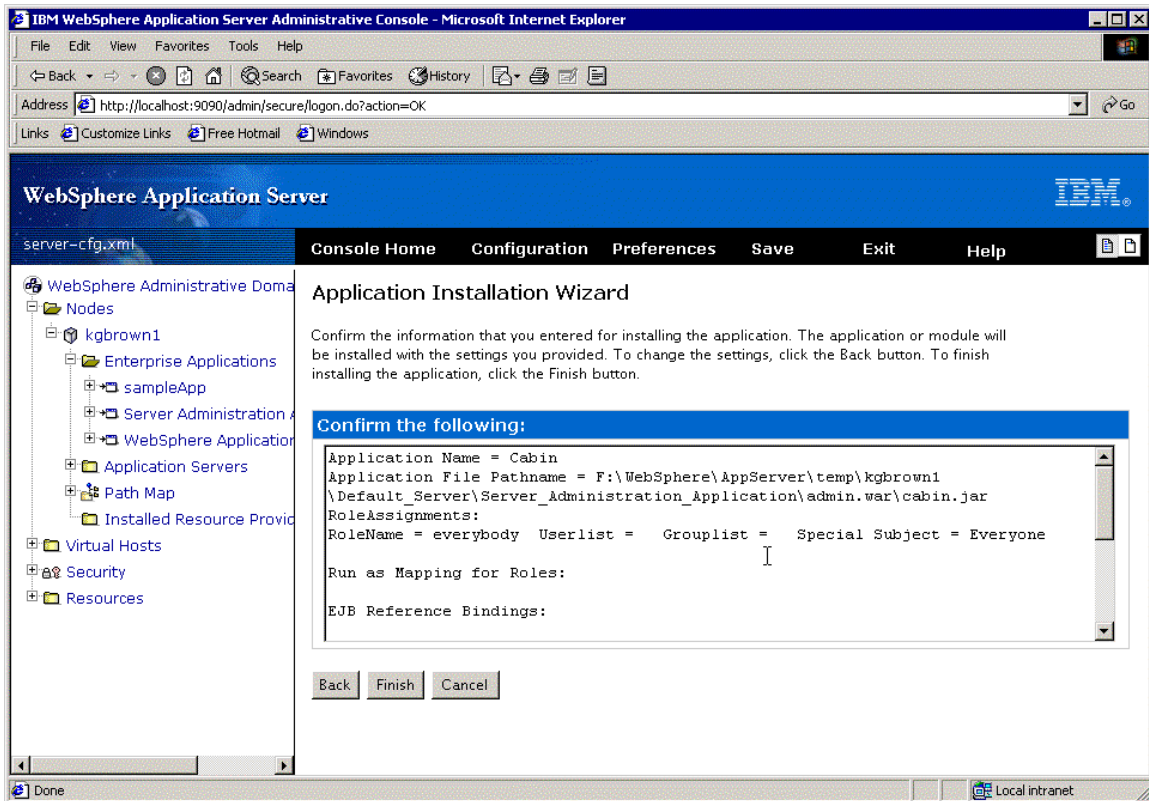
Enterprise Bean	JNDI Name	User ID	Password
cabin.jar:CabinEJB	<input type="text" value="jdbc/SampleDB"/>	<input type="text"/>	<input type="text"/>

At the bottom of the wizard are three buttons: "Back", "Next", and "Cancel". The "Next" button is highlighted with a mouse cursor.

This wizard page allows you to configure various deployment settings related to the CMP EJBs in your EJB *jar* file. As in the code generator built into the AAT, you can set the database type and schema name here. If you are using IBM DB2 7.2, and want to use the default schema name, you can leave these settings alone. If you are using a different database, you will need to change the database type here.

One setting you should configure, however, is the link to the data source used by the CMP. There are two ways to do so. Each EJB *jar* file contains a default data source reference. Likewise, each CMP EJB can be mapped to a specific data source, different from the default. Because we have only one EJB in this *jar* file, it doesn't matter which one we set. So, you can set either, or both, to be **jdbc/SampleDB**, to correspond to the DB2 data source you configured in the previous step. Then press **Next** to move on to the final page of this wizard:

Figure 51: Confirming settings



This page summarizes your settings and allows you a last chance to go back and change them. If all the settings correspond to the previous directions, then press the **Finish** button to deploy the EJB. Deployment may take several minutes; an animation will let you know it's still proceeding. At the end of the deployment process, you will see the following page in the right side of the Administrative Console:






Figure 52: Installed application list with Cabin application

 Configuration needs to be saved

Enterprise Applications

The J2EE applications (EAR files) installed on the application server

[For more information...](#)

<div> <div>Start</div> <div>Stop</div> <div>Restart</div> <div>Install</div> <div>Uninstall</div> <div>Export</div> <div>Export DDL</div> </div>		
<input type="checkbox"/>	Name	Archive URL
<input type="checkbox"/>	 sampleApp	\${APP_INSTALL_ROOT}/sampleApp.ear
<input type="checkbox"/>	 Server Administration Application	\${APP_INSTALL_ROOT}/admin.ear
<input type="checkbox"/>	 WebSphere Application Server Samples	\${APP_INSTALL_ROOT}\Samples.ear
<input type="checkbox"/>	 petstore	\${APP_INSTALL_ROOT}\petstore.ear
<input type="checkbox"/>	 Cabin	\${APP_INSTALL_ROOT}\Cabin.ear

Once again the link at the top of this list reminds you to save the configuration. Follow that link, as in the previous step, then exit the Administrative Console and stop the Application Server. Because you changed the XML configuration file in the process of installing the new application, the changes will not take effect until the server is restarted. After verifying that the application server has shut down, restart it as before, from the Windows Start menu or the First Steps application.

After the application server has started, verify that the standard output file contains the following line:

```

||| <timestamp> <id> EJBEngine      I WSVR0037I: Starting EJB jar:
    <path>Cabin.ear/cabin.jar

```

There should not be any errors after the EJB *jar* file has started. If your application server starts successfully, the Cabin EJB has successfully started inside it, and you are ready to move on to the final step in this exercise.

- ❖ Note: At this point you may be wondering why you didn't deploy the *.ear* file that you created earlier in the AAT. The answer is that you could have. We deployed the *.jar* file purely for didactic purposes, to show you that you can deploy either of the J2EE file types. Later you will see how to deploy *.ear* files instead of *.jar* files.

Step 9: Test your clients with the *launchClient* tool

You are almost done with the exercise! Take a deep breath, open a Windows command prompt, and take a moment to learn about the WebSphere *launchClient* tool.

launchClient is simply a *.bat* file that invokes a Java program that can in turn start J2EE application clients placed in deployed *.ear* files. It's easy to use. You simply type *launchClient* at the command prompt, followed by the name of a deployed *.ear* file. Other command-line options allow you to specify (for instance) which of multiple application client *.jar* files contained in an *.ear* file to use. If you invoke the *.bat* without any parameters it will display the list of options.

Note that *launchClient* is case-sensitive. If you see an error message stating it can't find the client *.jar* file, unzip your deployed application *.ear* file and verify that the capitalization of the client *.jar* file name on the command line is the same as it is in the *.ear* file.

In this case, cd to the directory where you placed the deployed *.ear* file and issue the command:

```
launchClient Deployed_Cabin.ear
```

You should see the following text, indicating that *Client_1* has successfully created a Cabin:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001

WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.cabin.Client_1
Master Suite
1
1
3
```

If you see an error message instead of this text, and you've verified that capitalization of the *.jar* file's name is not the problem, carefully go back over the previous steps to determine whether your EJB was correctly deployed, then restart the application server and try again.

You may have noticed that, even though you added two application client *.jar* files in Step 4 of the exercise, and you didn't specify which one of the two to start when you executed *launchClient*, *Client_1* ran and *Client_2* didn't. Why? When you specify only the *.ear* in a *launchClient* command, by default the first client added to the *.ear* file is the only one started. You can, however, indicate that you want to start any arbitrary application in the *.ear* file by specifying its *.jar* file name in a *-CC* command-line option.

Now, for the last part of this exercise, you will start *Client_2*. Type the following on the command line:

```
launchClient Deployed_Cabin.ear -CCjar=Client_2.jar
```

If things work correctly, you should see the following output displayed on the console:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001

WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has complet
ed.
WSCL0014I: Invoking the Application Client class
com.titan.cabin.Client_2
Trying to obtain initial context.
InitialContext obtained.
CabinHome found
Narrow finished -- proceeding to make cabins
PK = 1, Ship = 1, Deck = 1, BedCount = 3, Name = Master Suite
PK = 2, Ship = 1, Deck = 1, BedCount = 2, Name = Suite 100
PK = 3, Ship = 1, Deck = 1, BedCount = 3, Name = Suite 101
PK = 4, Ship = 1, Deck = 1, BedCount = 2, Name = Suite 102
PK = 5, Ship = 1, Deck = 1, BedCount = 3, Name = Suite 103
PK = 6, Ship = 1, Deck = 1, BedCount = 2, Name = Suite 104
PK = 7, Ship = 1, Deck = 1, BedCount = 3, Name = Suite 105
PK = 8, Ship = 1, Deck = 1, BedCount = 2, Name = Suite 106
PK = 9, Ship = 1, Deck = 1, BedCount = 3, Name = Suite 107
PK = 10, Ship = 1, Deck = 1, BedCount = 2, Name = Suite 108
PK = 11, Ship = 1, Deck = 2, BedCount = 2, Name = Suite 200
PK = 12, Ship = 1, Deck = 2, BedCount = 3, Name = Suite 201
PK = 13, Ship = 1, Deck = 2, BedCount = 2, Name = Suite 202
PK = 14, Ship = 1, Deck = 2, BedCount = 3, Name = Suite 203
PK = 15, Ship = 1, Deck = 2, BedCount = 2, Name = Suite 204
PK = 16, Ship = 1, Deck = 2, BedCount = 3, Name = Suite 205
PK = 17, Ship = 1, Deck = 2, BedCount = 2, Name = Suite 206
PK = 18, Ship = 1, Deck = 2, BedCount = 3, Name = Suite 207
PK = 19, Ship = 1, Deck = 2, BedCount = 2, Name = Suite 208
PK = 20, Ship = 1, Deck = 2, BedCount = 3, Name = Suite 209
PK = 21, Ship = 1, Deck = 3, BedCount = 2, Name = Suite 300
PK = 22, Ship = 1, Deck = 3, BedCount = 3, Name = Suite 301
PK = 23, Ship = 1, Deck = 3, BedCount = 2, Name = Suite 302
PK = 24, Ship = 1, Deck = 3, BedCount = 3, Name = Suite 303
PK = 25, Ship = 1, Deck = 3, BedCount = 2, Name = Suite 304
```

```
PK = 26, Ship = 1, Deck = 3, BedCount = 3, Name = Suite 305
PK = 27, Ship = 1, Deck = 3, BedCount = 2, Name = Suite 306
PK = 28, Ship = 1, Deck = 3, BedCount = 3, Name = Suite 307
PK = 29, Ship = 1, Deck = 3, BedCount = 2, Name = Suite 308
PK = 30, Ship = 1, Deck = 3, BedCount = 3, Name = Suite 309
PK = 31, Ship = 2, Deck = 1, BedCount = 2, Name = Suite 100
PK = 32, Ship = 2, Deck = 1, BedCount = 3, Name = Suite 101
PK = 33, Ship = 2, Deck = 1, BedCount = 2, Name = Suite 102
PK = 34, Ship = 2, Deck = 1, BedCount = 3, Name = Suite 103
PK = 35, Ship = 2, Deck = 1, BedCount = 2, Name = Suite 104
PK = 36, Ship = 2, Deck = 1, BedCount = 3, Name = Suite 105
PK = 37, Ship = 2, Deck = 1, BedCount = 2, Name = Suite 106
PK = 38, Ship = 2, Deck = 1, BedCount = 3, Name = Suite 107
PK = 39, Ship = 2, Deck = 1, BedCount = 2, Name = Suite 108
PK = 40, Ship = 2, Deck = 1, BedCount = 3, Name = Suite 109
PK = 41, Ship = 2, Deck = 2, BedCount = 2, Name = Suite 200
PK = 42, Ship = 2, Deck = 2, BedCount = 3, Name = Suite 201
PK = 43, Ship = 2, Deck = 2, BedCount = 2, Name = Suite 202
PK = 44, Ship = 2, Deck = 2, BedCount = 3, Name = Suite 203
PK = 45, Ship = 2, Deck = 2, BedCount = 2, Name = Suite 204
PK = 46, Ship = 2, Deck = 2, BedCount = 3, Name = Suite 205
PK = 47, Ship = 2, Deck = 2, BedCount = 2, Name = Suite 206
PK = 48, Ship = 2, Deck = 2, BedCount = 3, Name = Suite 207
PK = 49, Ship = 2, Deck = 2, BedCount = 2, Name = Suite 208
PK = 50, Ship = 2, Deck = 2, BedCount = 3, Name = Suite 209
PK = 51, Ship = 2, Deck = 3, BedCount = 2, Name = Suite 300
PK = 52, Ship = 2, Deck = 3, BedCount = 3, Name = Suite 301
PK = 53, Ship = 2, Deck = 3, BedCount = 2, Name = Suite 302
PK = 54, Ship = 2, Deck = 3, BedCount = 3, Name = Suite 303
PK = 55, Ship = 2, Deck = 3, BedCount = 2, Name = Suite 304
PK = 56, Ship = 2, Deck = 3, BedCount = 3, Name = Suite 305
PK = 57, Ship = 2, Deck = 3, BedCount = 2, Name = Suite 306
PK = 58, Ship = 2, Deck = 3, BedCount = 3, Name = Suite 307
PK = 59, Ship = 2, Deck = 3, BedCount = 2, Name = Suite 308
PK = 60, Ship = 2, Deck = 3, BedCount = 3, Name = Suite 309
PK = 61, Ship = 3, Deck = 1, BedCount = 2, Name = Suite 100
PK = 62, Ship = 3, Deck = 1, BedCount = 3, Name = Suite 101
PK = 63, Ship = 3, Deck = 1, BedCount = 2, Name = Suite 102
PK = 64, Ship = 3, Deck = 1, BedCount = 3, Name = Suite 103
PK = 65, Ship = 3, Deck = 1, BedCount = 2, Name = Suite 104
PK = 66, Ship = 3, Deck = 1, BedCount = 3, Name = Suite 105
PK = 67, Ship = 3, Deck = 1, BedCount = 2, Name = Suite 106
PK = 68, Ship = 3, Deck = 1, BedCount = 3, Name = Suite 107
PK = 69, Ship = 3, Deck = 1, BedCount = 2, Name = Suite 108
PK = 70, Ship = 3, Deck = 1, BedCount = 3, Name = Suite 109
PK = 71, Ship = 3, Deck = 2, BedCount = 2, Name = Suite 200
```

```
PK = 72, Ship = 3, Deck = 2, BedCount = 3, Name = Suite 201
PK = 73, Ship = 3, Deck = 2, BedCount = 2, Name = Suite 202
PK = 74, Ship = 3, Deck = 2, BedCount = 3, Name = Suite 203
PK = 75, Ship = 3, Deck = 2, BedCount = 2, Name = Suite 204
PK = 76, Ship = 3, Deck = 2, BedCount = 3, Name = Suite 205
PK = 77, Ship = 3, Deck = 2, BedCount = 2, Name = Suite 206
PK = 78, Ship = 3, Deck = 2, BedCount = 3, Name = Suite 207
PK = 79, Ship = 3, Deck = 2, BedCount = 2, Name = Suite 208
PK = 80, Ship = 3, Deck = 2, BedCount = 3, Name = Suite 209
PK = 81, Ship = 3, Deck = 3, BedCount = 2, Name = Suite 300
PK = 82, Ship = 3, Deck = 3, BedCount = 3, Name = Suite 301
PK = 83, Ship = 3, Deck = 3, BedCount = 2, Name = Suite 302
PK = 84, Ship = 3, Deck = 3, BedCount = 3, Name = Suite 303
PK = 85, Ship = 3, Deck = 3, BedCount = 2, Name = Suite 304
PK = 86, Ship = 3, Deck = 3, BedCount = 3, Name = Suite 305
PK = 87, Ship = 3, Deck = 3, BedCount = 2, Name = Suite 306
PK = 88, Ship = 3, Deck = 3, BedCount = 3, Name = Suite 307
PK = 89, Ship = 3, Deck = 3, BedCount = 2, Name = Suite 308
PK = 90, Ship = 3, Deck = 3, BedCount = 3, Name = Suite 309
PK = 91, Ship = 3, Deck = 4, BedCount = 2, Name = Suite 400
PK = 92, Ship = 3, Deck = 4, BedCount = 3, Name = Suite 401
PK = 93, Ship = 3, Deck = 4, BedCount = 2, Name = Suite 402
PK = 94, Ship = 3, Deck = 4, BedCount = 3, Name = Suite 403
PK = 95, Ship = 3, Deck = 4, BedCount = 2, Name = Suite 404
PK = 96, Ship = 3, Deck = 4, BedCount = 3, Name = Suite 405
PK = 97, Ship = 3, Deck = 4, BedCount = 2, Name = Suite 406
PK = 98, Ship = 3, Deck = 4, BedCount = 3, Name = Suite 407
PK = 99, Ship = 3, Deck = 4, BedCount = 2, Name = Suite 408
PK = 100, Ship = 3, Deck = 4, BedCount = 3, Name = Suite 409
```

One quick thing to note before you finish this section: The two test clients defined in this exercise both change the persistent state of the database. Because they insert rows into the CabinEJB table, you cannot successfully run either client a second time without first deleting the rows that were inserted into the table. To empty the table, open a DB2 CLP, connect to the [SAMPLE](#) database, and type:

```
Delete from CabinEJB
```

Finally, you will want to be certain that you have inserted the rows in the table before you run the *Client_3* EJB client that is defined in the next section. To check, in the DB2 CLP enter:

```
Select * from CabinEJB
```

If this command returns `0 row(s) selected`, re-run the two client programs, but if it returns `100 row(s) selected`, you are ready to move on.

Exercise 4.2: A Simple Session Bean

In this exercise you will create and deploy the TravelAgent session bean, then test it. The EJB book contains both the source code for the bean and the *Client_3* test program.

To accomplish these tasks, you will:

1. Compile the TravelAgent classes and use the AAT to build an EJB *.jar* file for the TravelAgent bean
2. Create a client *.jar* for the *Client_3* test program
3. Add the client *.jar* file and the new EJB *.jar* file to the *.ear* file created in the previous exercise
4. Deploy the EJB *.jar* file to IBM WebSphere Application Server, AEs
5. Run the *Client_3* test program with the *launchClient* program

Step 1: Compile the TravelAgent bean classes and build an EJB .jar file

You will begin this step by compiling the classes for the TravelAgent example, including the three *.java* files that make up the TravelAgent EJB and *Client_3.java*. A batch file like the one you used to compile the Cabin EJB classes will help:

```
<InstallDrive>/EJBWorkbook/Source/Chapter4/dev/TravelAgentCompile.bat
```

To execute this batch file, open a command prompt, change to the *dev* directory, and then type **TravelAgentCompile** at the prompt. After the batch file finishes, verify that you now have *.class* files for all of the *.java* files in the *com/titan/travelagent* directory.

Now you can move on to learning another way to use the AAT: to create an EJB *.jar* file interactively rather than building it from scratch with command-line tools. As before, launch the AAT from the First Steps tool, but this time at the Startup screen select **Create EJB Module Wizard** and press **OK**.

At this point you will see a page similar to ones you've already seen in the AAT:

Figure 53: Specifying EJB module properties

Create EJB Module Wizard

Specifying EJB Module Properties

Enter a display name and file name for your EJB module. The display name is used to identify the EJB module.

Select a parent application from the list, or leave blank to create a stand-alone module. Enter the file name of your EJB module. EJB modules have a .jar extension.

Containing application: Browse...

Display name:

File name: *

Description:

Help < Back Next > Finish Cancel

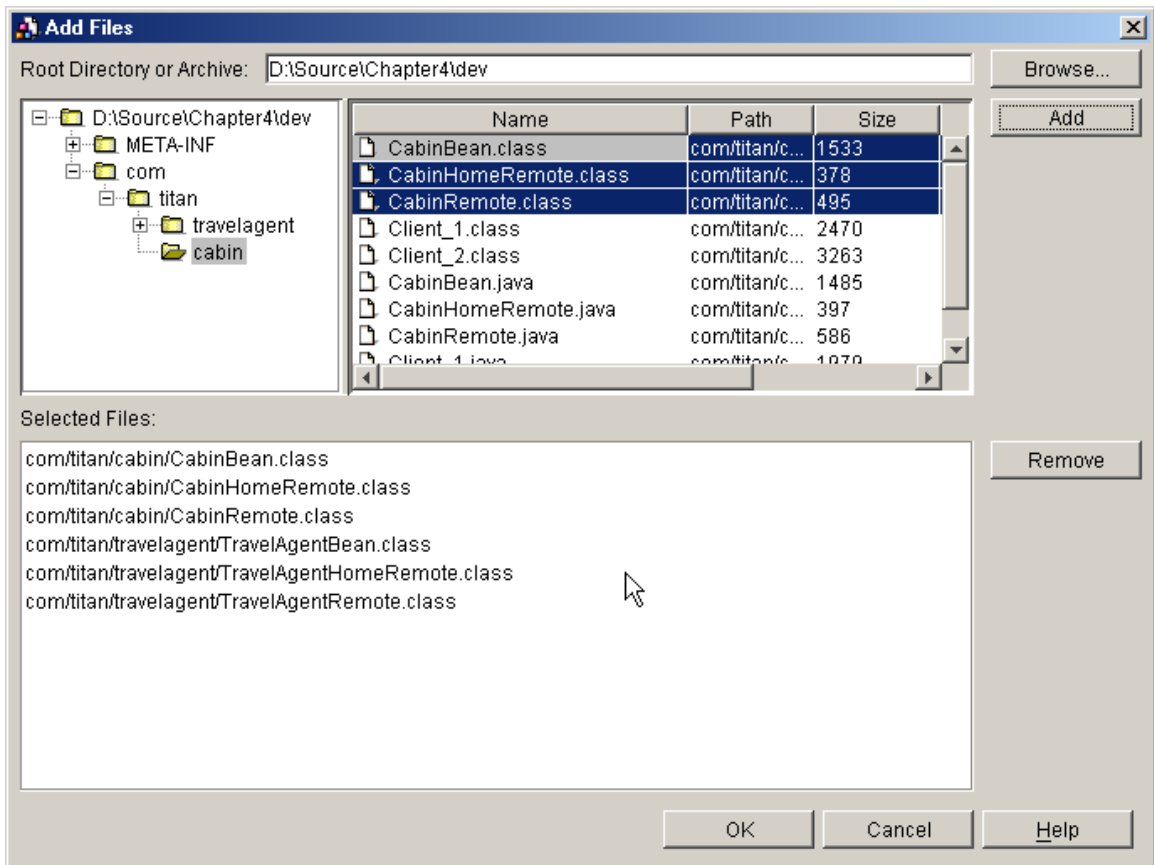
In this page you want to enter the following information:

- ◆ **Display Name:** **TravelAgentBean**
- ◆ **File Name:** **TravelAgent.jar**
- ◆ **Description:** Any string you choose

Be sure to leave the **Containing Application** field blank. Later you will add this EJB module to the .ear file you have been building manually.

Press the **Next** button to bring you to an Add Files wizard page exactly like the ones you have seen before. In this case, though, you will be adding a few more files than before. Press the **Add...** button to bring up the following dialog:

Figure 54: Adding files for the TravelAgent Bean



Use the **Browse...** button to get to the appropriate subdirectory of *dev*. To add the TravelAgent bean correctly you need to add a few more files than you may think. First of all, you must add the bean's home interface, remote interface, and bean implementation classes:

com/titan/travelagent/TravelAgentBean.class

com/titan/travelagent/TravelAgentHomeRemote.class

com/titan/travelagent/TravelAgentRemote.class

But you can't stop there. Remember from the EJB book that the TravelAgent session bean refers to the Cabin entity bean. That means that the compiled code for the TravelAgent bean must be able to use the code for the Cabin EJB's home and remote interfaces, so be sure you add these classes as well:

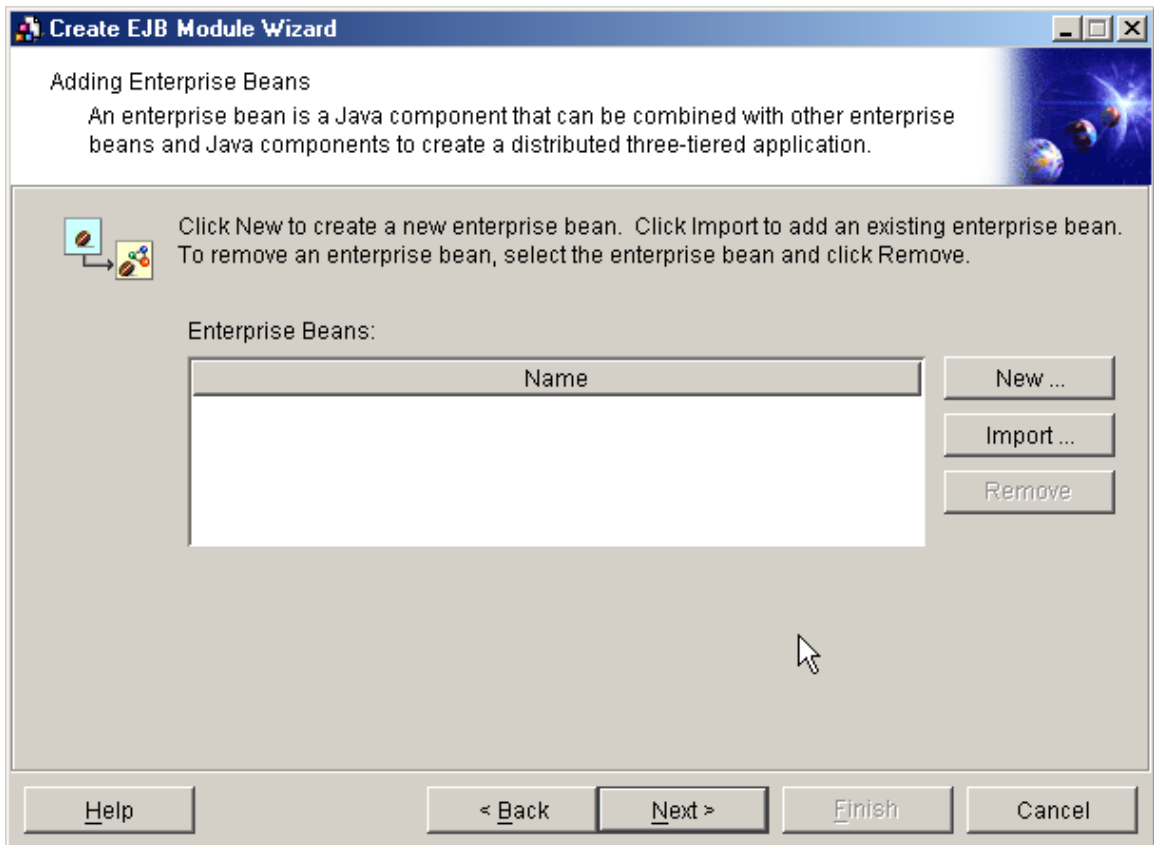
`com/titan/cabin/CabinHomeRemote.class`

`com/titan/cabin/CabinRemote.class`

After all these files have been added, press **OK** to dismiss this dialog. The Add Files wizard page should now contain all of the files you have added. Move on to the next wizard page by pressing the **Next** button.

The next page in this wizard enables you to add EJB client application files. You will do that separately, so press **Next** to move past that page, and on to the one for specifying EJB module icons. You won't specify any new icons for your module, so press **Next** again to move on to the Adding Enterprise Beans page:

Figure 55: Adding enterprise beans



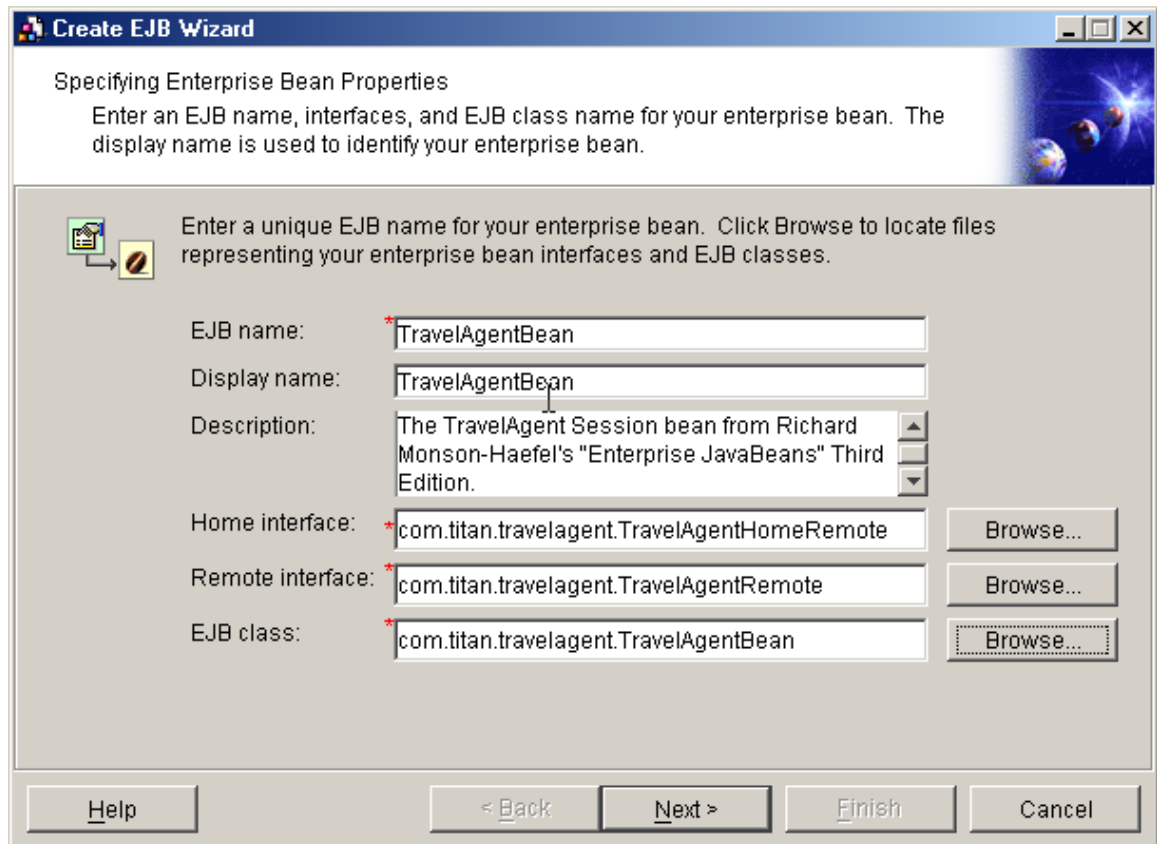
You are now ready to add your TravelAgent bean to this EJB *.jar* file, so press the **New...** button. The first dialog allows you to specify the bean type:

Figure 56: Choosing the bean type



You want to build a session bean, so accept the default selection and press **OK**. The following dialog (which is also a wizard itself) will appear:

Figure 57: Specifying EJB properties

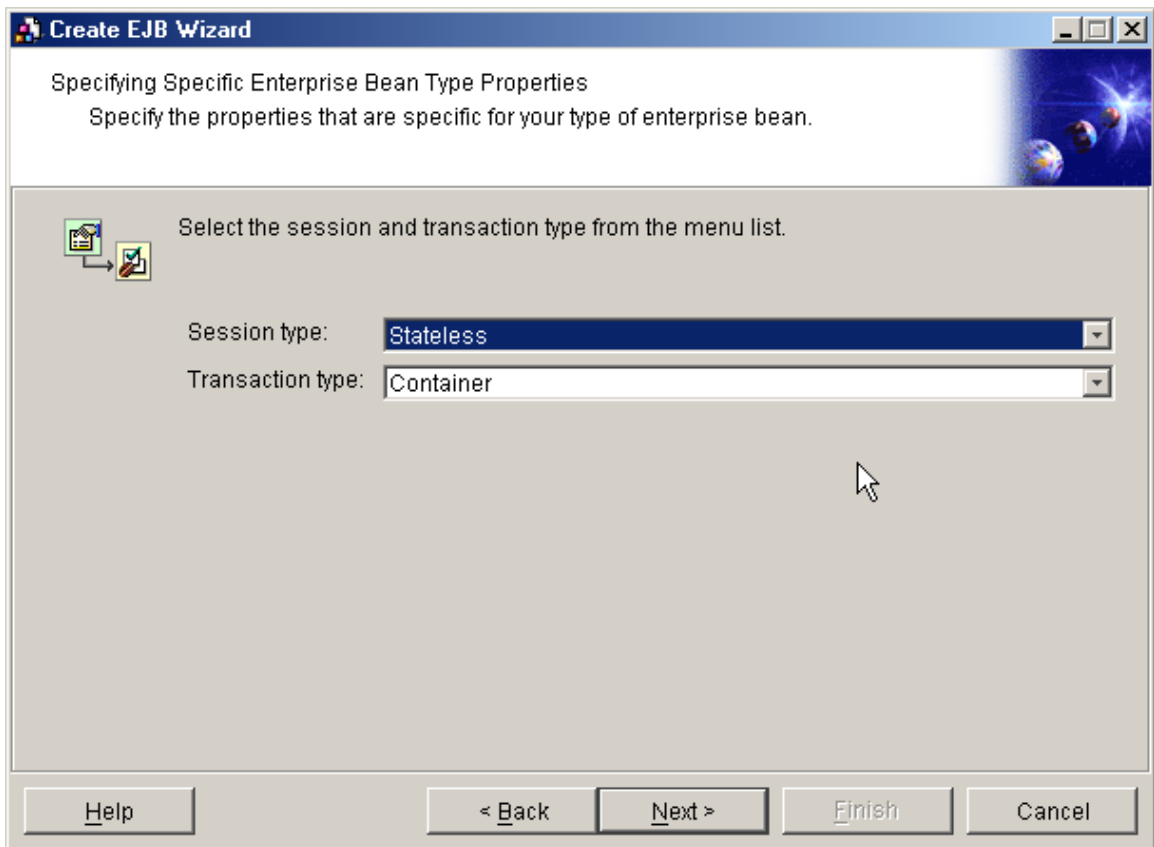


Here you will specify the component parts of the TravelAgent bean. Enter the following information into this dialog:

- ◆ **EJB Name:** TravelAgentBean
- ◆ **Display Name:** TravelAgentBean
- ◆ **Description:** Any descriptive string you like
- ◆ **Home Interface:** Use **Browse...** to select *com.titan.travelagent.TravelAgentHomeRemote*
- ◆ **Remote Interface:** Use **Browse...** to select *com.titan.travelagent.TravelAgentRemote*
- ◆ **EJB Class:** Use **Browse...** to select *com.titan.travelagent.TravelAgentBean*

Now press **Next** to move on to the next wizard page:

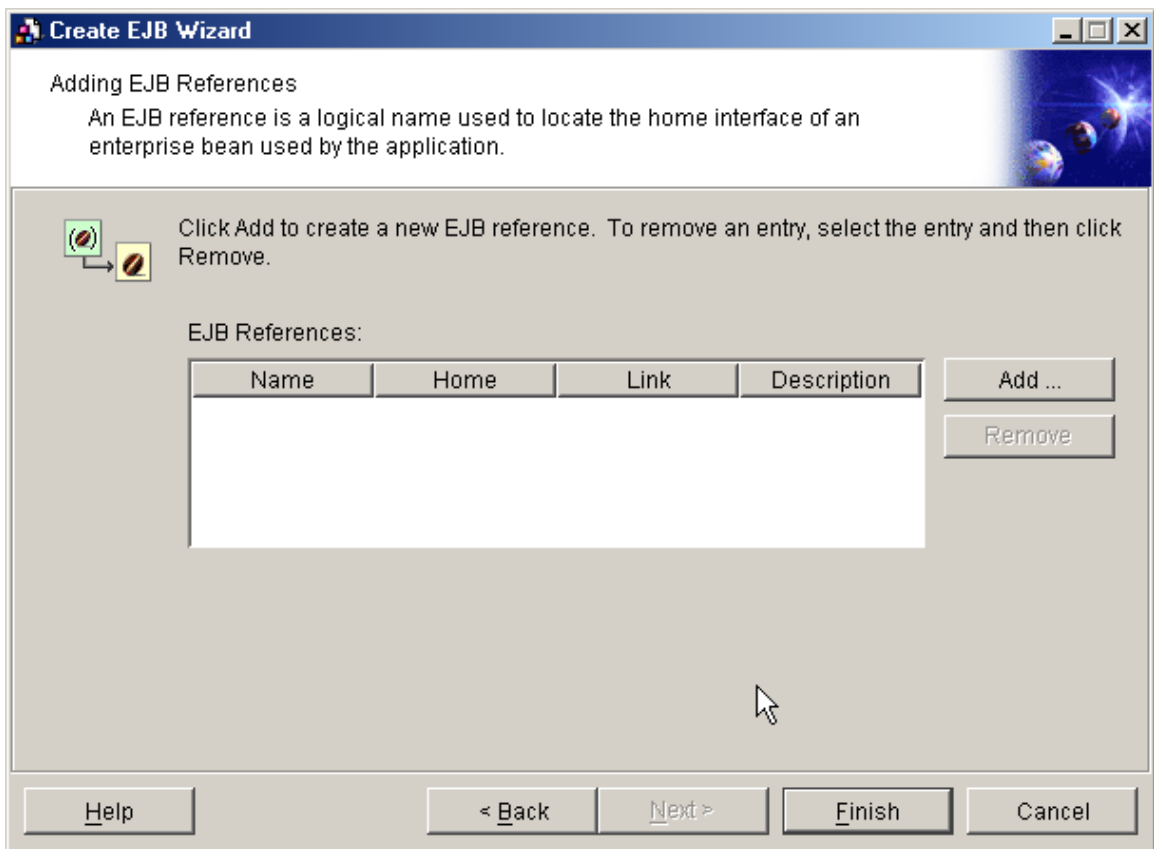
Figure 58: Specifying EJB type properties



This page allows you to specify characteristics of the EJB that are unique to the bean type (session or entity). In this case, you want to build a stateless session bean, so pick **Stateless** in the **Session Type** drop-down. Leave the selection in the **Transaction Type** drop-down as **Container** (our EJB will not use Bean-Managed Transactions, but will instead use Container Managed Transactions as described in the EJB Book) and press **Next** to move on.

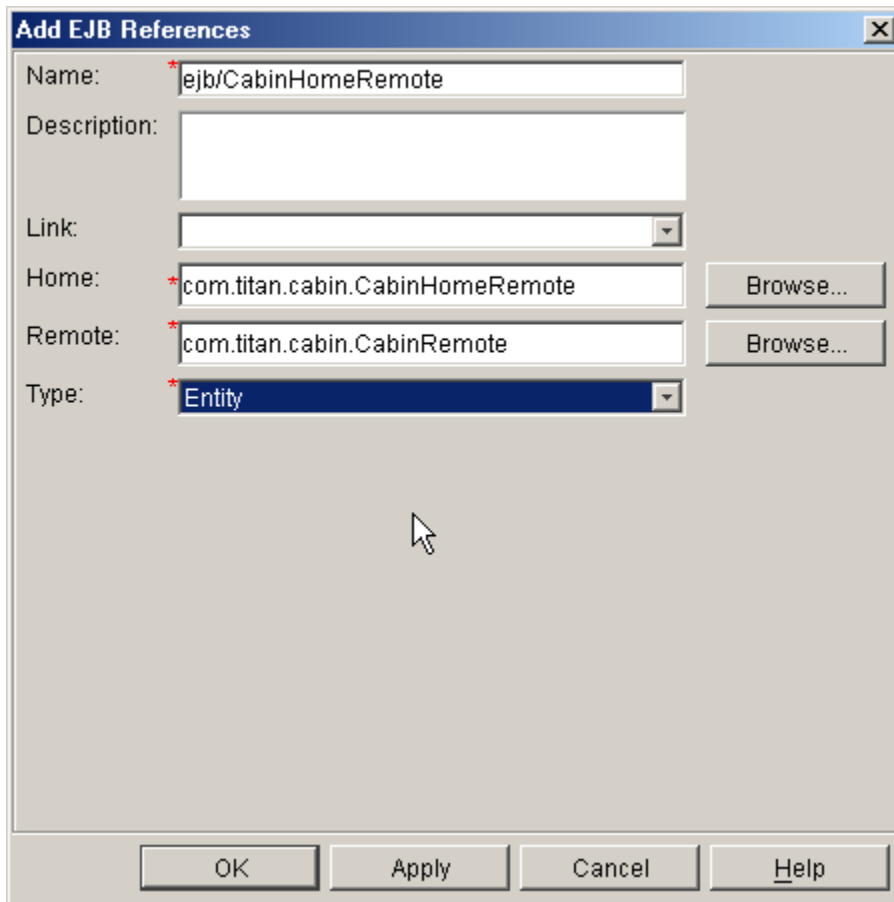
At this point you will see a page for entering icons. Press **Next**. Then you will see a page for environment entries. Press **Next** again. Then you will see a page for entering security role references; you do not yet have any, so press **Next**. The next page allows you to specify resource references; your session bean is not using any external resources, so press **Next** again to get to the following dialog:

Figure 59: Adding EJB references – empty



You do need to interact with this wizard page, because your TravelAgent session EJB does refer to another EJB by an abstract reference, *ejb/CabinHome*. Press **Add...** to see how to make this abstract reference concrete:

Figure 60: Adding an EJB reference

The image shows a Java Swing dialog box titled "Add EJB References". It contains several input fields and buttons. The "Name:" field is filled with "ejb/CabinHomeRemote". The "Description:" field is empty. The "Link:" field is empty with a dropdown arrow. The "Home:" field is filled with "com.titan.cabin.CabinHomeRemote" and has a "Browse..." button to its right. The "Remote:" field is filled with "com.titan.cabin.CabinRemote" and has a "Browse..." button to its right. The "Type:" field is a dropdown menu with "Entity" selected. At the bottom are "OK", "Apply", "Cancel", and "Help" buttons. A mouse cursor is visible over the dialog.

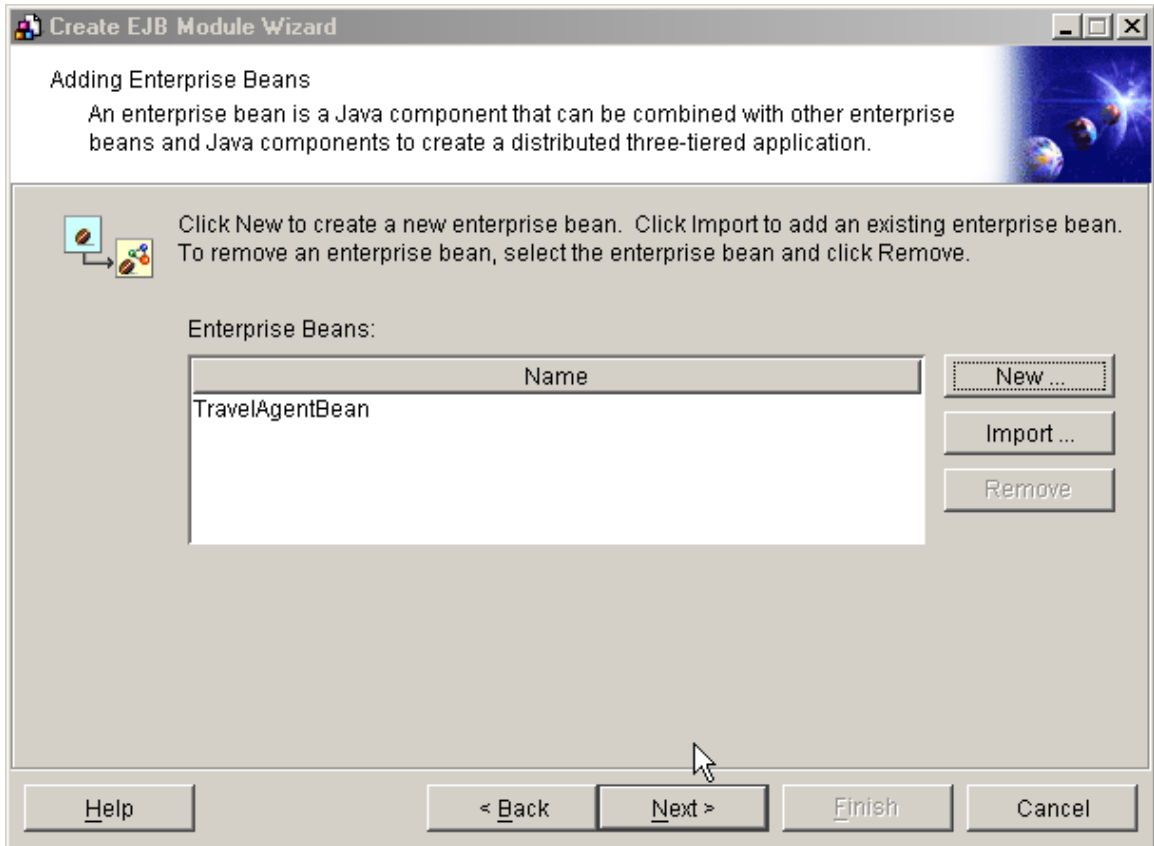
This dialog allows you to specify the Cabin information that the TravelAgent bean refers to abstractly. Enter the following information:

- ◆ **Name:** `ejb/CabinHomeRemote`
- ◆ **Description:** Any string you like
- ◆ **Home:** Use **Browse...** to select `com.titan.cabin.CabinHomeRemote`
- ◆ **Remote:** Use **Browse...** to select `com.titan.cabin.CabinRemote`
- ◆ **Type:** Select **Entity** from the drop-down

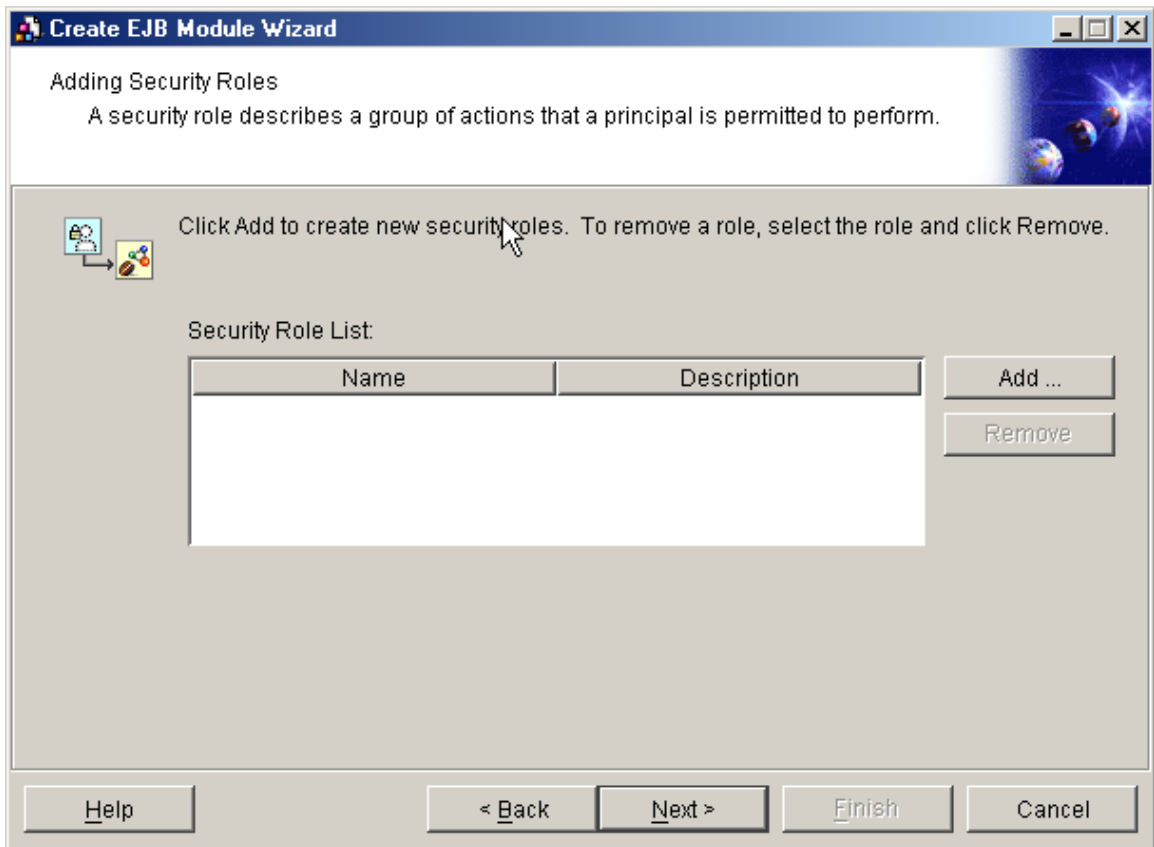
Be sure to leave the **Link** field empty.

Press **OK** to dismiss this dialog, and then press **Finish** to complete the Create EJB wizard. Next, verify that the EJB has been successfully added to the list of EJBs in the wizard page:

Figure 61: Adding enterprise beans – TravelAgent added

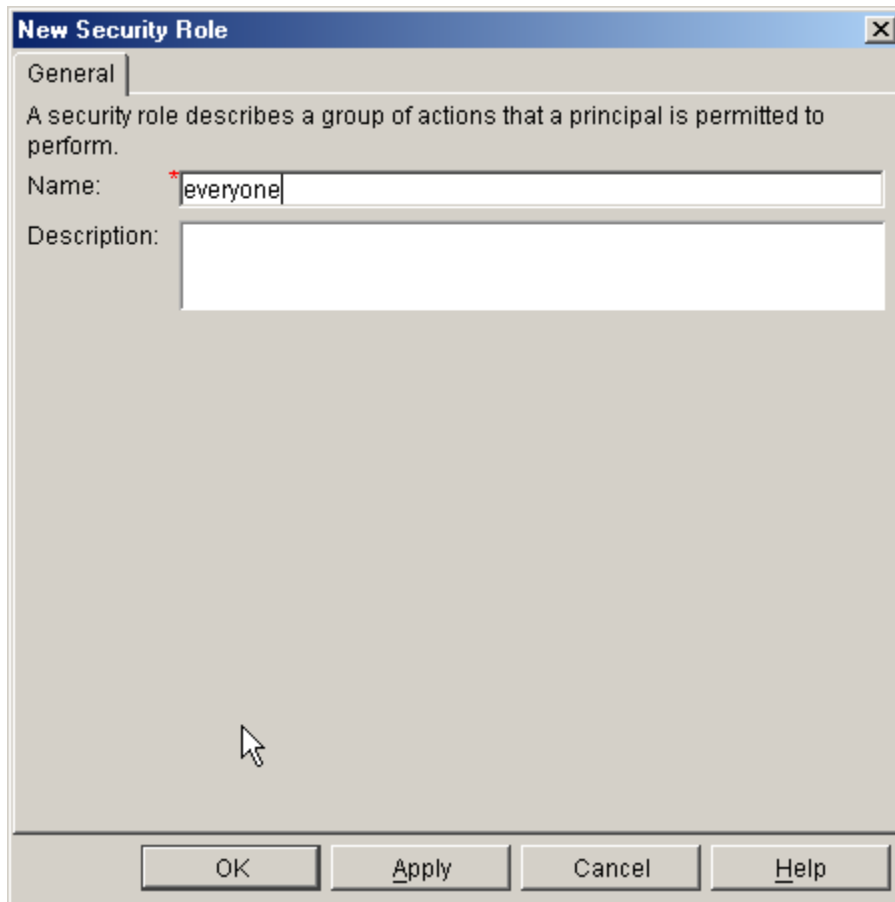


Now press the **Next** button to move on to the following page:

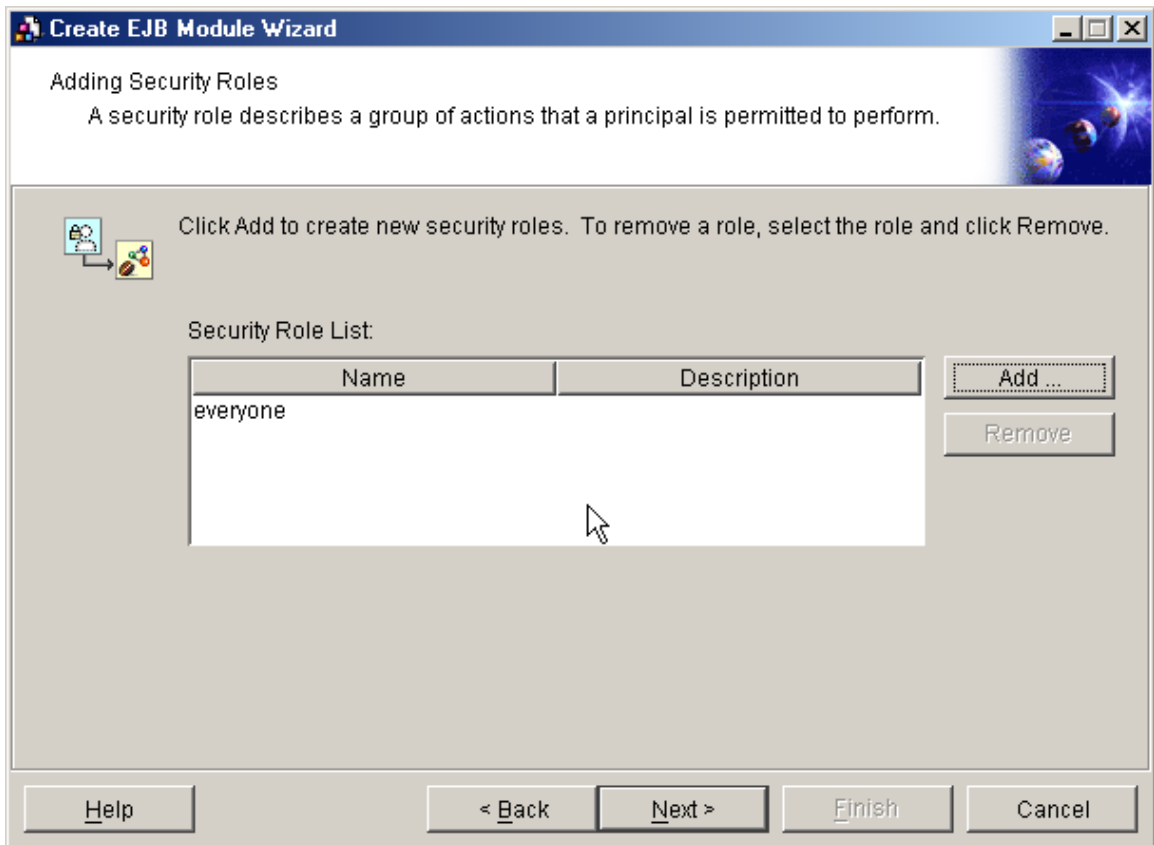
Figure 62: Adding new security roles

This wizard page allows you to add new security roles to the EJB *jar* file. You will need to add the “everyone” role that was described in the EJB book, so press the **Add...** button to display the following dialog:

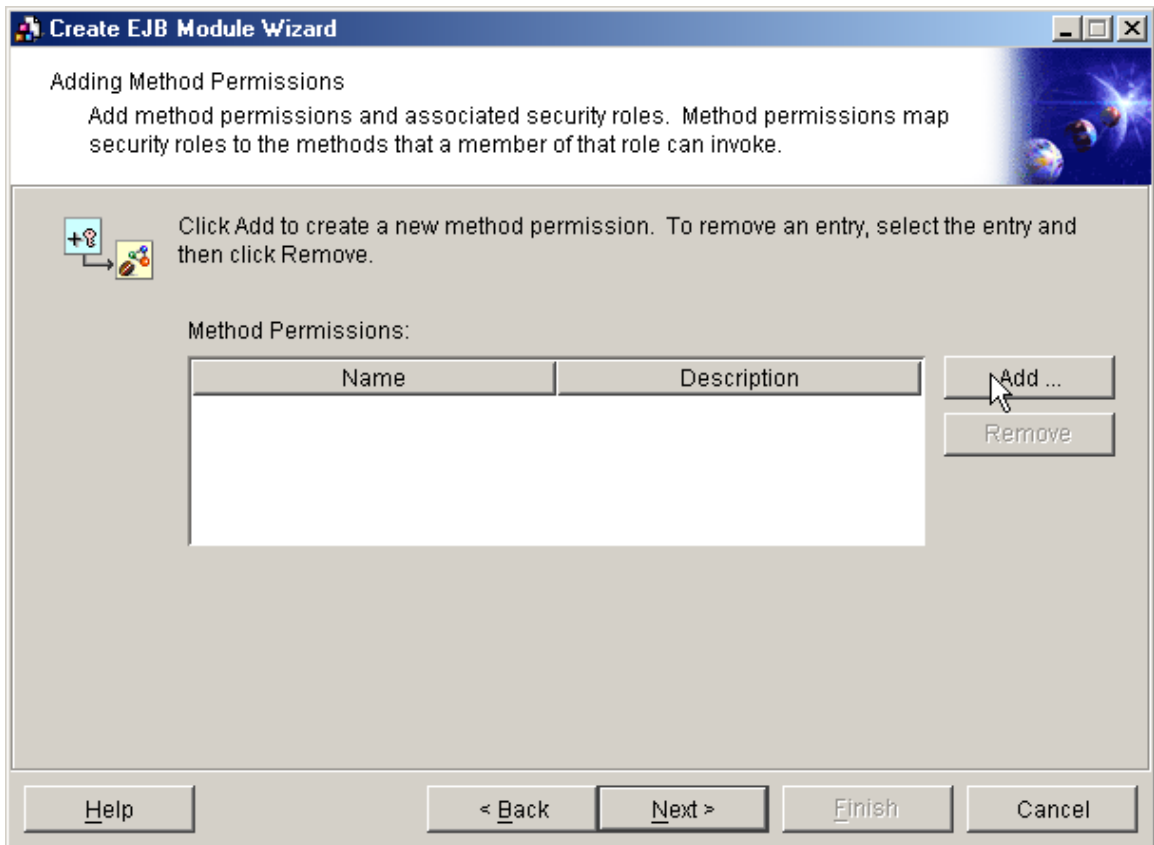
Figure 63: Creating a security role



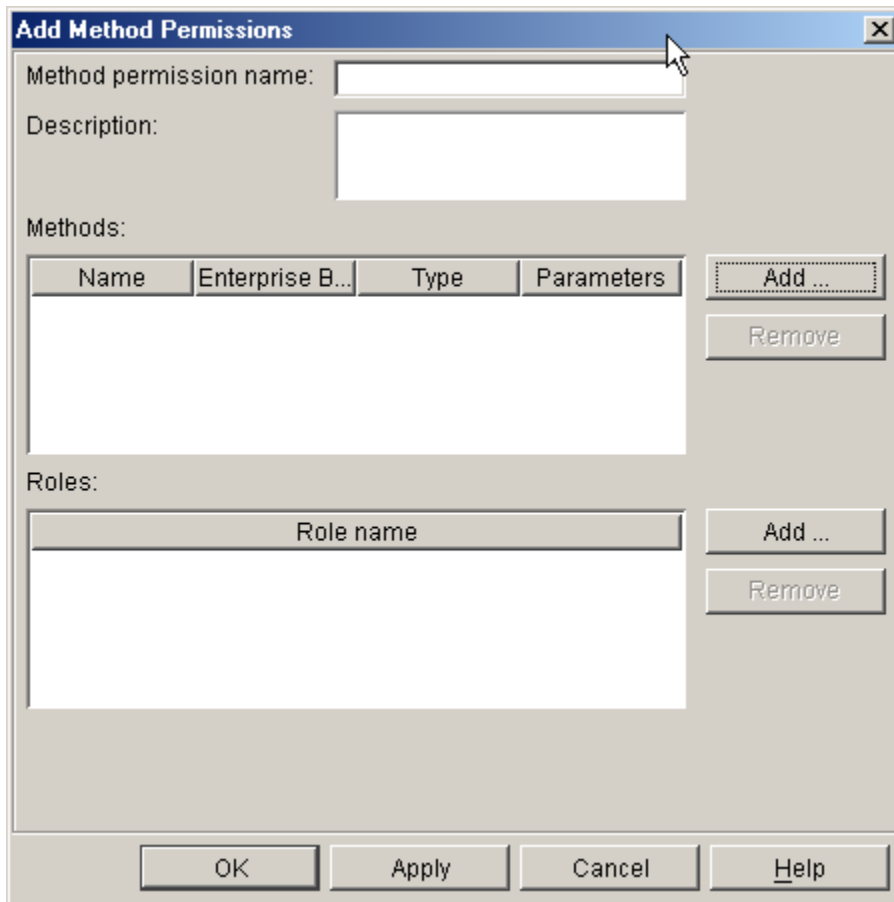
Here just type **everyone** in the **Name** field and press the **OK** button. Now the security roles dialog will appear as it does here:

Figure 64: New security role added

You have finished adding security roles, so press the **Next** button to define how this security role will be used in this EJB.

Figure 65: Adding method permissions

This wizard page displays your bean's current method permissions and allows you to add more. Press the **Add...** button to display the following dialog:

Figure 66: Adding a method permission

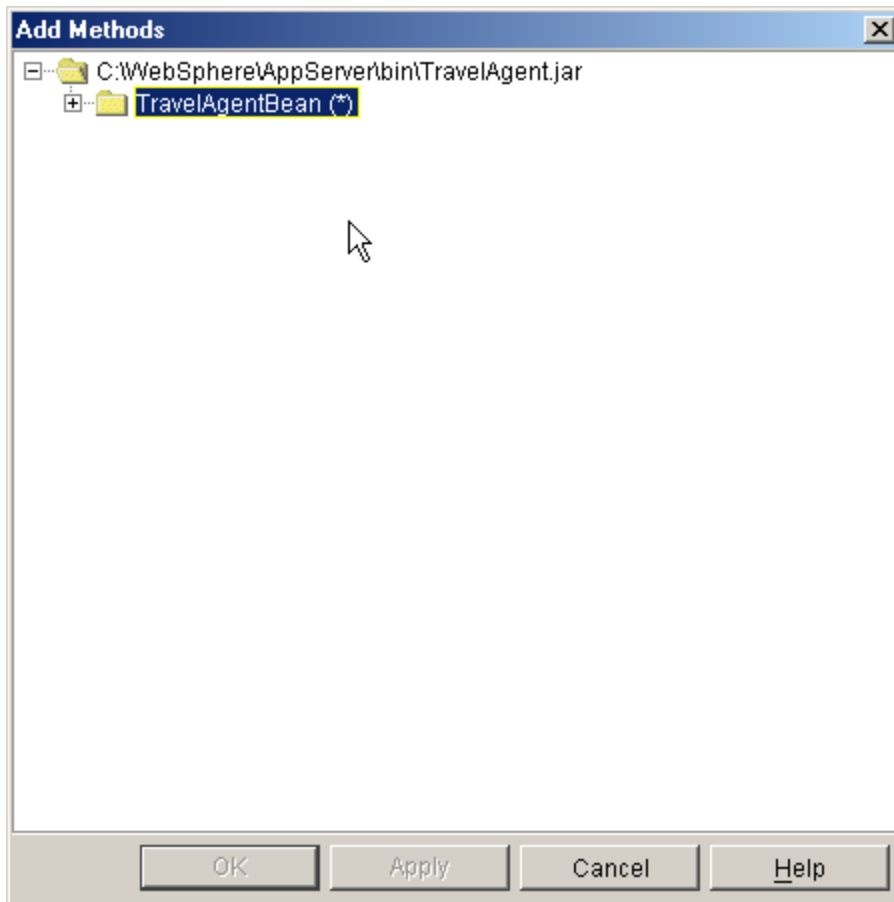
The dialog box titled "Add Method Permissions" contains the following fields and controls:

- Method permission name:** A text input field.
- Description:** A text input field.
- Methods:** A table with columns: Name, Enterprise B..., Type, and Parameters. To the right of the table are buttons "Add ..." and "Remove".
- Roles:** A table with a single column: Role name. To the right of the table are buttons "Add ..." and "Remove".
- Buttons:** OK, Apply, Cancel, and Help at the bottom.

Name	Enterprise B...	Type	Parameters
------	-----------------	------	------------

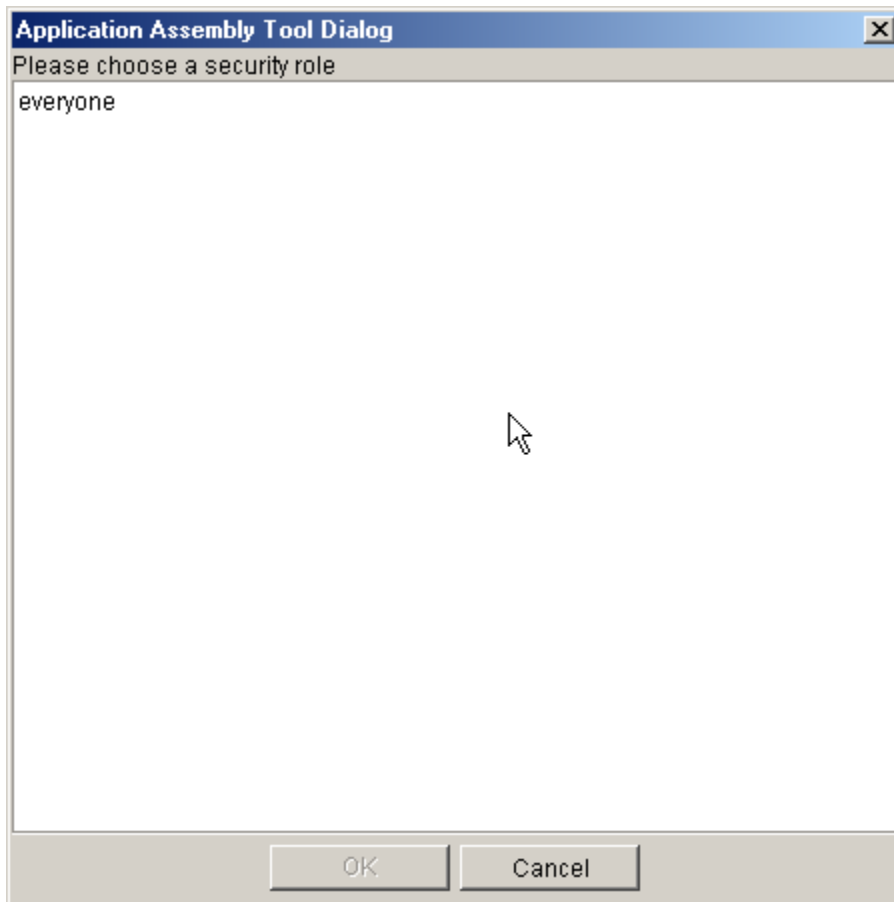
Role name

Here you can define links between the methods of the TravelAgent EJB and the roles you defined earlier – in other words, to specify who can invoke which of the bean's methods. You will begin by adding methods to the list in the upper table. Press the **Add...** button next to that table to bring up the following dialog:

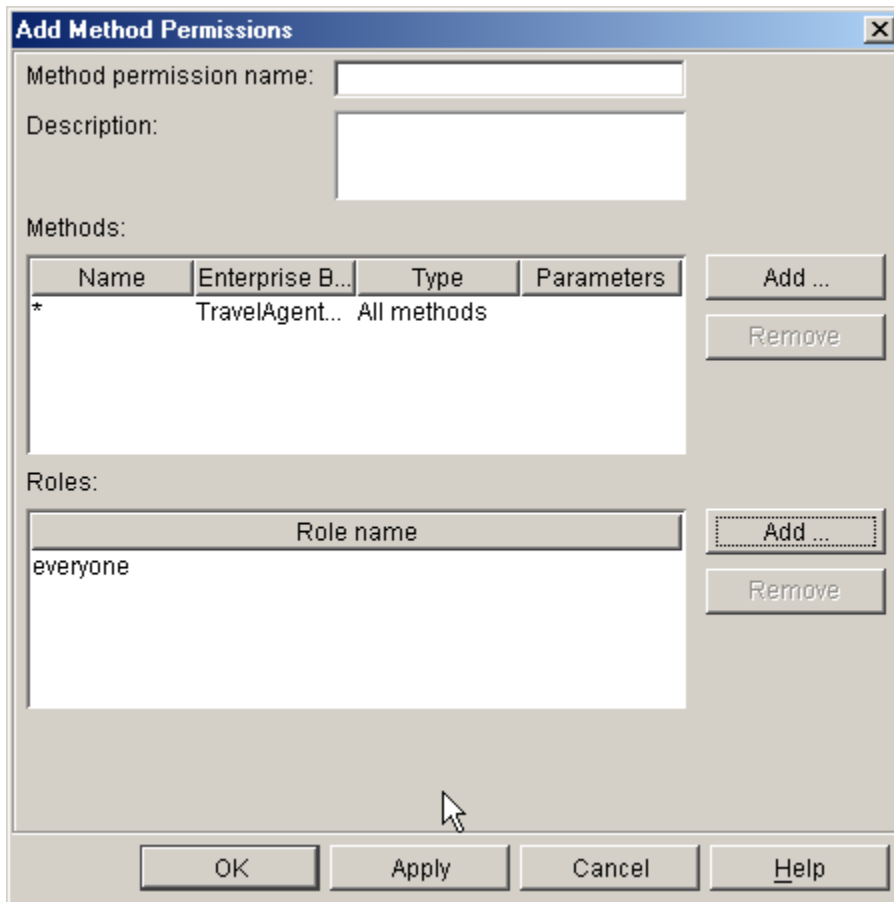
Figure 67: Adding methods

Most of this dialog is a tree view, which you can expand to show all the methods of the home and remote interfaces of your EJB. In this case, you want to assign the same role permissions to all of the methods in your EJB's Home and Remote interfaces, so you need only expand the tree one level, to *TravelAgentBean (*)*. Select the second level of the tree and press the **OK** button to return to the previous wizard page. There, press the lower **Add...** button to display the following dialog:

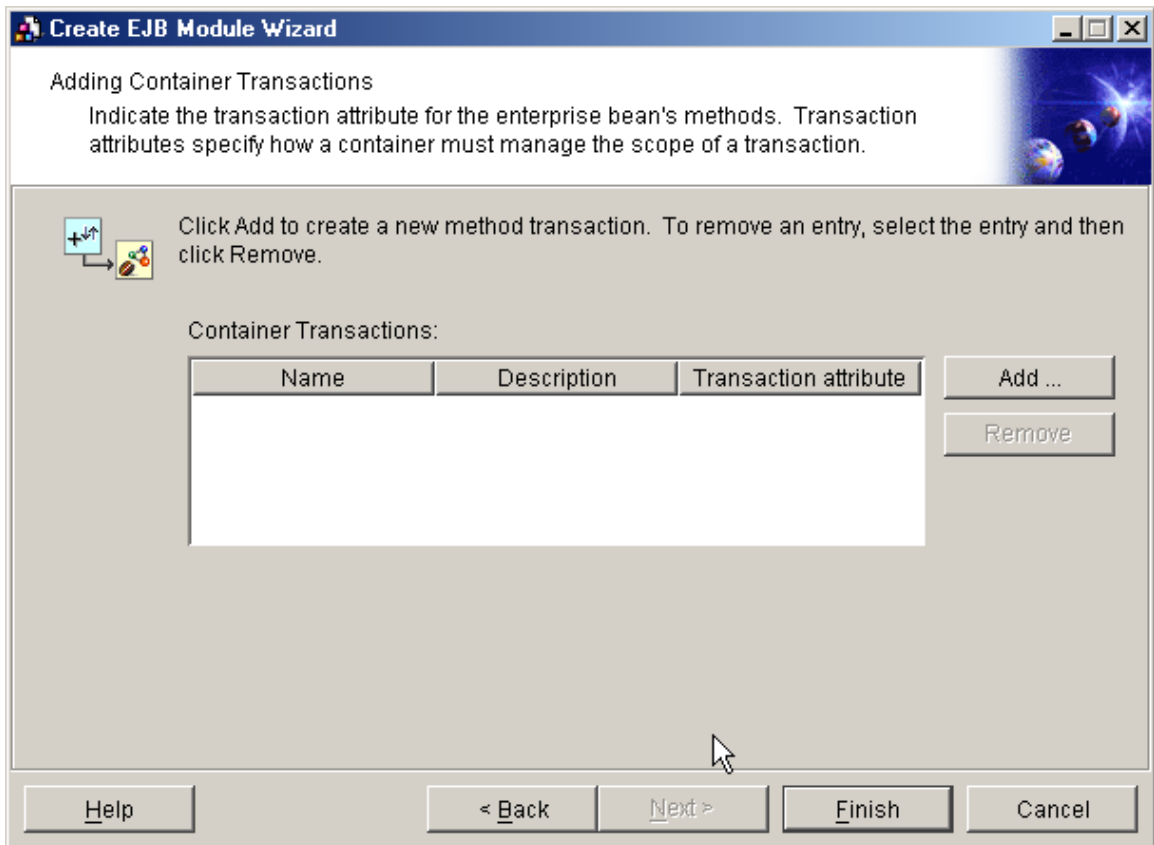
Figure 68: Selecting a security role



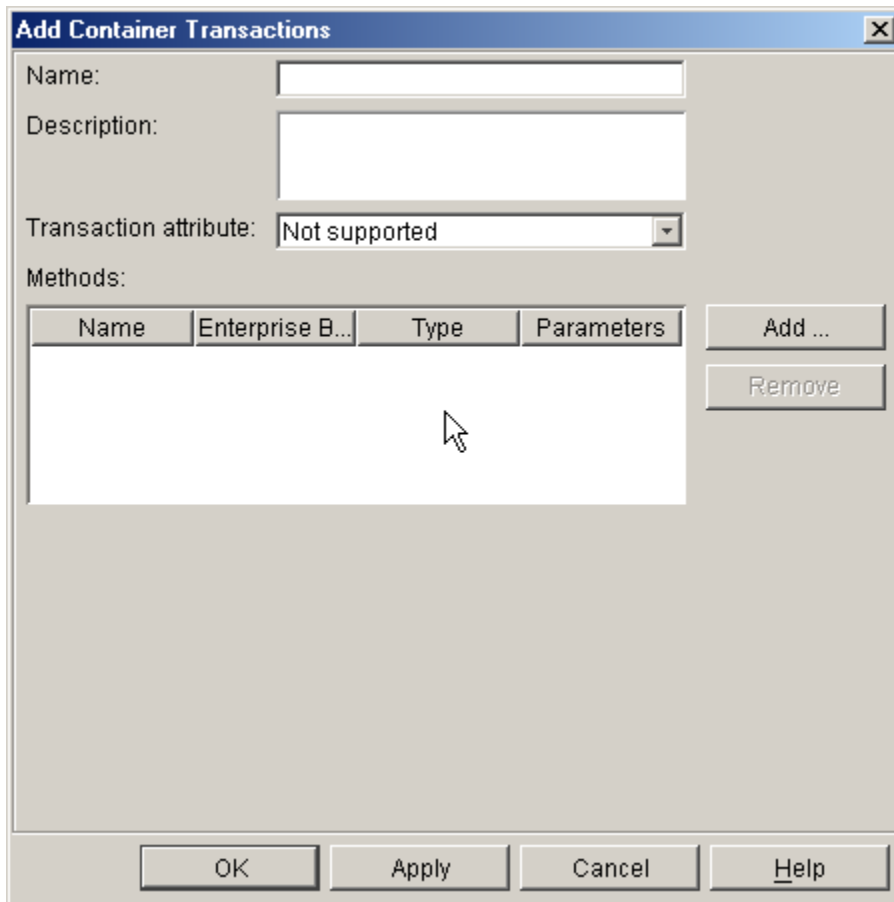
Only the single “everyone” role defined earlier appears. Select it and press **OK** to return once more to the Add Method Permissions dialog, where you'll see the methods and roles you specified:

Figure 69: Adding method permissions – completed

Press the **OK** button to dismiss this dialog and return to the wizard page for adding method permissions. Here you will see that the new method permission (seen as `<name not specified>`) has been added to the list. Press the **Next** button to move on to the final wizard page:

Figure 70: Adding container transactions

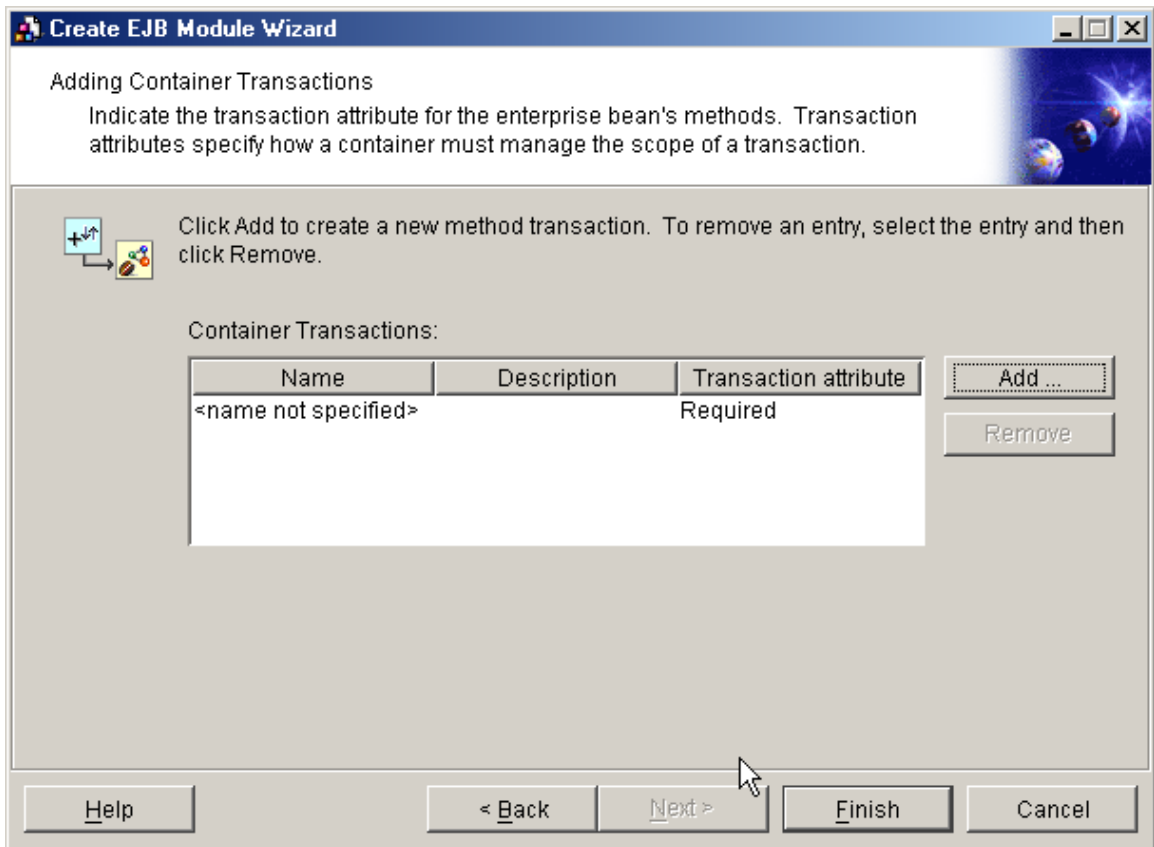
Here you will specify the transaction attributes for your bean's methods. Press the **Add...** button to begin this process by displaying the dialog below:

Figure 71: Adding a container transaction

This dialog lets you specify each method's transaction attributes individually if you need to, but in this case you'll specify the same attribute for all of them. Begin by changing the **Transaction Attribute** drop-down selection from **Not Supported** to **Required**. Then press the **Add...** button. You'll see a method selection tree exactly like the one you saw a few moments ago. As you did then, expand the method tree out to the second level (*TravelAgentBean (*)*), select that element, and press **OK** to dismiss the method selection dialog.

The Add Container Transactions dialog's methods table should now show a listing for "all methods" of your EJB. Once you're satisfied that you've specified the TravelAgent bean's transaction attributes correctly, press **OK** to dismiss the dialog and return to the Adding Container Transactions wizard page:

Figure 72: Adding container transactions – completed



Except to note that the page displays the transaction attribute you specified, you are now finished with this wizard. Press the **Finish** button and you will be returned to the main window of the AAT. In the tree view there you should be able to expand the **EJB Modules** folder and see your new EJB. To finish this step, select **File→Save As...** and save your new EJB *.jar* file as *travelagent.jar* in the *dev* directory.

Step 2: Create an application client *.jar* file for the *Client_3* test program

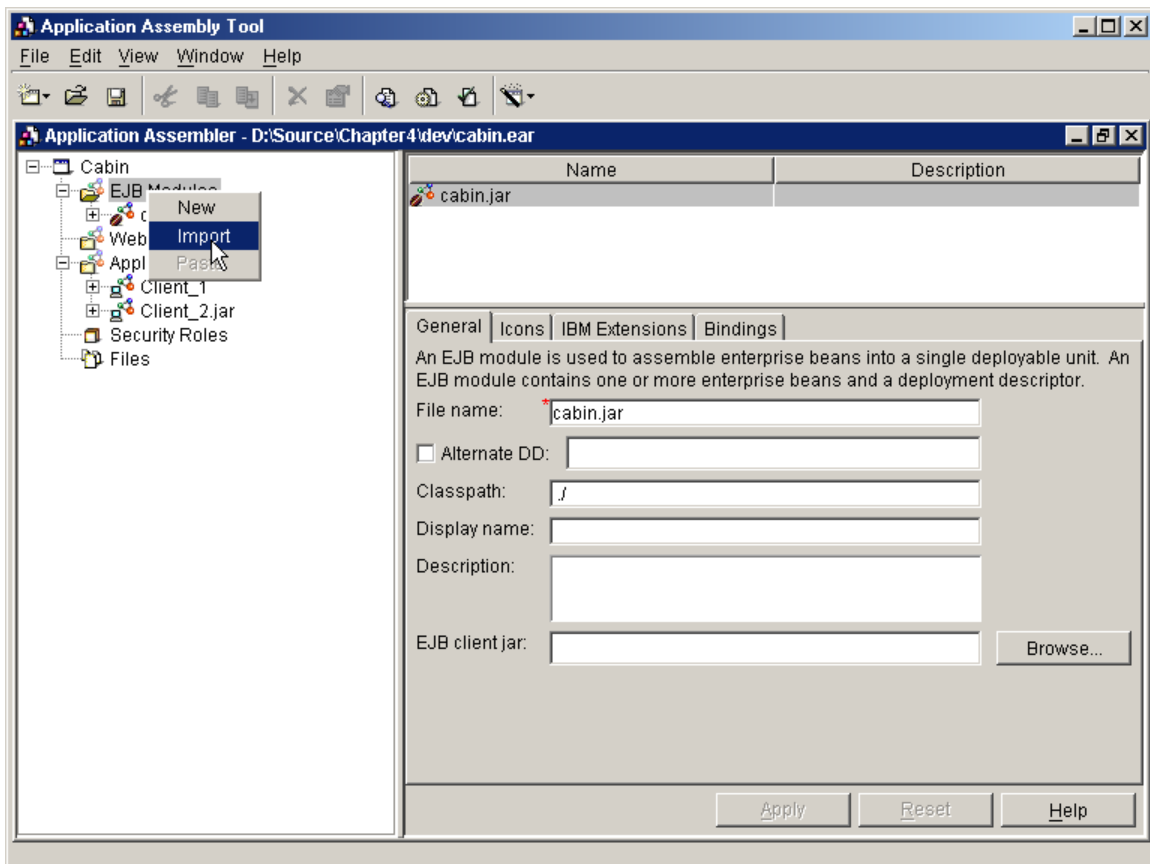
In Exercise 4.1 you learned how to use the AAT wizards to create an application client *.jar* file. Follow the same procedures to build such a *.jar* for the *Client_3* application. The key information is:

- ◆ **File Name:** `Client_3.jar`
- ◆ **Display Name:** `Client_3`
- ◆ **Main class** (which must first be added as a file to the `.jar` file):
`com.titan.travelagent.Client_3.class`

Step 3: Add the application client and the EJB to the .ear file

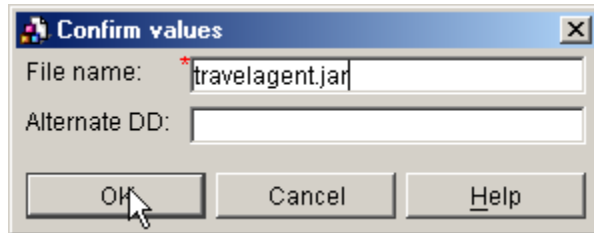
In this step you will examine the last major part of the AAT that you haven't investigated: – its ability to import existing J2EE components into an `.ear` file. Open the `cabin.ear` file using **File→Open...** When the `cabin.ear` file is displayed, select the **EJB Modules** folder, open its context menu (click the folder name with the right mouse button), and select **Import** as shown:

Figure 73: Importing the EJB module



You will see a standard Windows file dialog. Select the new *travelagent.jar* file you created in the previous step and press the **Open** button. When you do, the following dialog will appear:

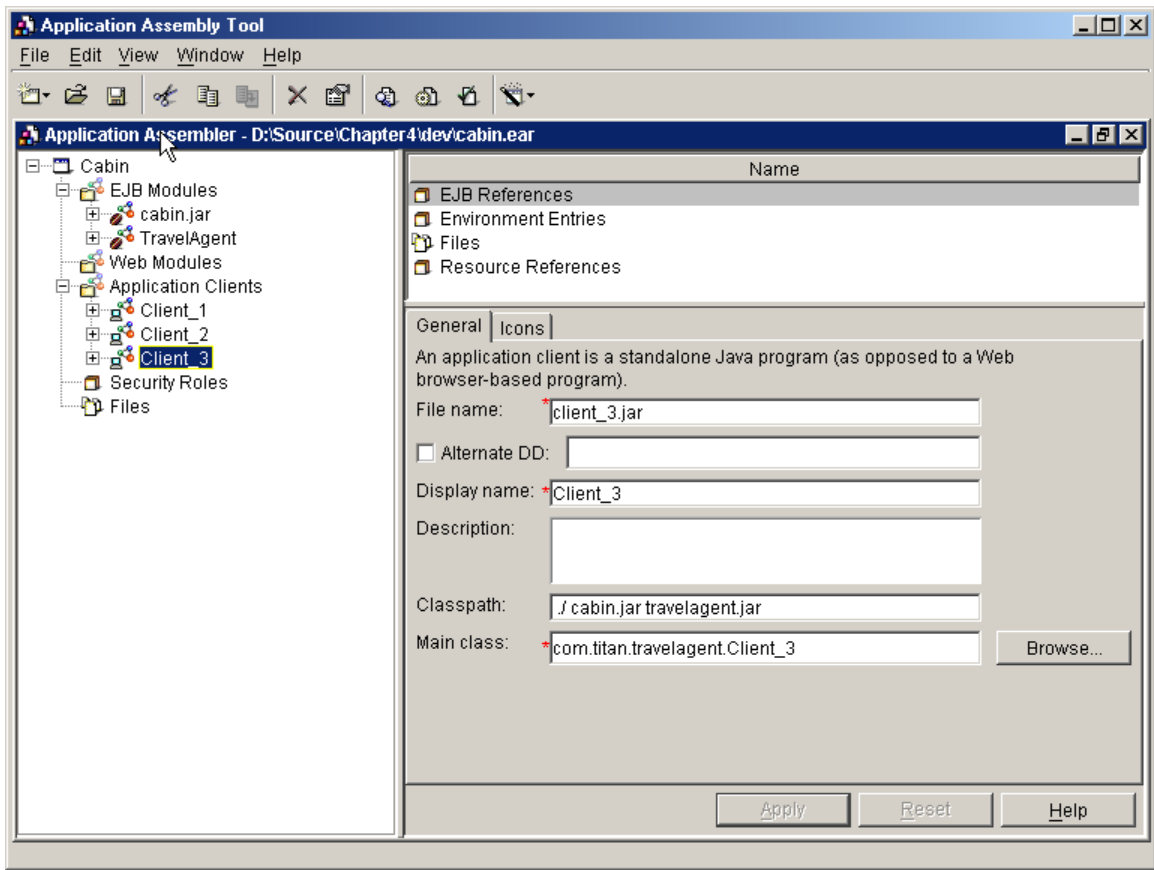
Figure 74: Confirming the TravelAgent deployment descriptor



This dialog allows you to confirm that you really want to add TravelAgent's *.jar* file to *cabin.ear* file. If you wanted to substitute an alternate deployment descriptor you could do so here. In this case, simply press the **OK** button to dismiss this dialog.

The AAT will now add the TravelAgent bean to the list of EJB modules. Follow the procedure outlined earlier to add the new *Client_3* application client *.jar* file – but this time select the **Application Clients** folder and open the context menu there to perform the import operation. When you have finished, you should see all three clients and both EJBs in the *.ear* file:

Figure 75: All clients and EJB modules added



Save the modified `.ear` file (using **File→Save**), then regenerate the deployment code for the `.ear` file, using **File→Generate Code For Deployment** as you did in Exercise 4.1.

Step 4: Deploy the EJB .jar file to the server

Exercise 4.1 detailed the procedure for using the Administrative Console to add the Cabin bean's EJB `.jar` file to the WebSphere Application Server. Follow those same steps to add the `travelagent.jar` file as a new enterprise application. Earlier you entered the JNDI name of your Cabin EJB as **CabinHomeRemote**; when you reach that same point this time make the JNDI name of your TravelAgent EJB **TravelAgentHomeRemote**.

You will encounter one additional wizard page you didn't see earlier:

Figure 76: Mapping *ejb/CabinHomeRemote* to a JNDI name

Application Installation Wizard

Each EJB Reference in your application or module must be mapped to an enterprise bean. For each EJB reference below, enter the name of an enterprise bean.

Mapping EJB References to Enterprise Beans		
Archives	References	Enterprise Bean JNDI Names
TravelAgentBean:TravelAgentBean	ejb/CabinHomeRemote	CabinHomeRemote

Back Next Cancel

Here you can specify the actual JNDI name of the EJB referred to in the TravelAgent EJB *jar* file as *ejb/CabinHomeRemote*. Enter **CabinHomeRemote** in the text field and press **Next**.

After completing the deployment of the new TravelAgent EJB, close the Administrative Console, stop the application server, and restart it. (Remember to verify that the server has completely stopped before restarting it, and then verify that the restart does not produce error messages in the standard output.)

Step 5: Run the *Client_3* test program

Now you should be ready to test the *Client_3* program against your new TravelAgent bean. At the command prompt type:

```
launchClient Deployed_Cabin.ear -CCjar=Client_3.jar
```

If all has gone well, you should see the following:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001
```

```
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.travelagent.Client_3
1,Master Suite,1
3,Suite 101,1
5,Suite 103,1
7,Suite 105,1
9,Suite 107,1
12,Suite 201,2
14,Suite 203,2
16,Suite 205,2
18,Suite 207,2
20,Suite 209,2
22,Suite 301,3
24,Suite 303,3
26,Suite 305,3
28,Suite 307,3
30,Suite 309,3
```

- ❖ Note: There was one odd little step that we took in this exercise that isn't exactly a "best practice" for deploying J2EE applications. In this exercise we directly included parts of the Cabin EJB (its home and remote interfaces) in the EJB *.jar* file of the TravelAgent EJB that referenced it – purely to simplify the steps of the exercise, and also to conform to the EJB *.jar* files that were described in the EJB book. As you can imagine, this approach could lead to problems in source code management in a real project, as the multiple copies of the files could diverge as changes are made over time. A better approach would have been to have included both EJBs in a single EJB *.jar* file, which would have been deployed to the server as a single unit. In this way we can treat the two EJBs as a single compilation unit (under the same set of controls) and catch problems at compilation time rather than deployment time.

Exercises for Chapter 5

Exercise 5.1: The Remote Component Interfaces

In this exercise you will compile and deploy several additional client programs as J2EE application *.jar* files and test them against the TravelAgent and Cabin EJBs deployed in earlier exercises.

To accomplish these tasks, you will:

1. Compile the Java source code for the client programs into *.class* files
2. Create new application client *.jar* files
3. Create a new *.ear* file containing the TravelAgent and Cabin EJBs and the new application client *.jar* files and generate deployment code for this *.ear* file.
4. Test your new application clients

Step 1: Compile the classes

Open a Windows command prompt and `cd` to this directory:

```
<InstallDrive>/EJBWorkbook/Source/Chapter5/Exercise5-1/dev
```

Once there, run the *Exercise5_1Compile.bat* batch file to compile the classes for this exercise. Before moving on to the next step, verify that there were no compilation errors, and that there are *.class* files for all of the *.java* files in the *Exercise5-1* directory structure.

Step 2: Create three new application client .jar files

In this step you will use the AAT to create three new application client *.jar* files. In each case, you can follow the steps for using the **Create Application Client** wizard you encountered in exercise 4.1, except to substitute the information below for the corresponding information in that exercise.

Begin by creating an application client for the *Client_51* class. Here is the information to enter in the wizard:

- ◆ **FileName:** *Client_51.jar*
- ◆ **Display Name:** *Client_51*
- ◆ **Files to import:** *com.titan.clients.Client_51.class* (in the file import dialog select the *Exercise5-1/dev* directory and browse the file structure)
- ◆ **Main Class:** *com.titan.clients.Client_51.class*
- ◆ No icons, EJB references, resource references, or environment entries

After completing the wizard, save the file as *Exercise5-1/dev/Client_51.jar*. Next, create an application client *jar* file for the *Client_51_undo* class. Again, use the **Create Application Client** wizard, this time entering:

- ◆ **FileName:** *Client_51_undo.jar*
- ◆ **Display Name:** *Client_51_undo*
- ◆ **Files to import:** *com.titan.clients.Client_51_undo.class* (in the file import dialog select the *Exercise5-1/dev* directory)
- ◆ **Main Class:** *com.titan.clients.Client_51_undo.class*
- ◆ No icons, EJB references, resource references, or environment entries

When you have finished, save the file as *Exercise5-1/dev/Client_51_undo.jar*. Finally, run the same wizard a third time and enter:

- ◆ **FileName:** *Client_52.jar*
- ◆ **Display Name:** *Client_52*
- ◆ **Files to import:** *com.titan.clients.Client_52.class* (in the file import dialog select the *Exercise5-1/dev* directory)
- ◆ **Main Class:** *com.titan.clients.Client_52.class*
- ◆ No icons, EJB references, resource references, or environment entries

Save the file as *Exercise5-1/dev/Client_52.jar* and move on to the next step.

Step 3: Create the Client_51.ear file

Using the AAT, assemble the three application clients, together with *travelagent.jar* and *cabin.jar*, to create an *.ear* file. When you are finished, save this new *.ear* file as *Exercise5-1/dev/Client_51.ear*. Finally, use the AAT's **Generate Deployed Code** menu option to generate the deployed code as you have done in previous exercises.

Step 4: Test the application clients

Ensure that the application server is running and that *Exercise5-1/dev* is the current directory, then type:

```
launchClient Deployed_Client_51.ear -CCjar=Client_51.jar
```

Validate the output in the console against the following:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.clients.Client_51
1st List: Before deleting cabin number 30
1,Master Suite,1
3,Suite 101,1
5,Suite 103,1
7,Suite 105,1
9,Suite 107,1
12,Suite 201,2
14,Suite 203,2
16,Suite 205,2
18,Suite 207,2
20,Suite 209,2
22,Suite 301,3
24,Suite 303,3
26,Suite 305,3
28,Suite 307,3
30,Suite 309,3
2nd List: After deleting cabin number 30
1,Master Suite,1
3,Suite 101,1
5,Suite 103,1
7,Suite 105,1
9,Suite 107,1
12,Suite 201,2
14,Suite 203,2
16,Suite 205,2
18,Suite 207,2
20,Suite 209,2
22,Suite 301,3
24,Suite 303,3
26,Suite 305,3
28,Suite 307,3
```

Next run *Client_51_undo*:

```
launchClient Deployed_Client_51.ear -CCjar=Client_51_undo.jar
```

Verify the output against:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.clients.Client_51_undo
1st List: Before re-creating cabin number 30
1,Master Suite,1
3,Suite 101,1
5,Suite 103,1
7,Suite 105,1
9,Suite 107,1
12,Suite 201,2
14,Suite 203,2
16,Suite 205,2
18,Suite 207,2
20,Suite 209,2
22,Suite 301,3
24,Suite 303,3
26,Suite 305,3
28,Suite 307,3
2nd List: After re-creating cabin number 30
1,Master Suite,1
3,Suite 101,1
5,Suite 103,1
7,Suite 105,1
9,Suite 107,1
12,Suite 201,2
14,Suite 203,2
16,Suite 205,2
18,Suite 207,2
20,Suite 209,2
22,Suite 301,3
24,Suite 303,3
26,Suite 305,3
28,Suite 307,3
30,Suite 309,3
```

Finally, run *Client_52*:

```
launchClient Deployed_Client_51.ear -CCjar=Client_52.jar
```

Verify its output against:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.clients.Client_52
com.titan.cabin.CabinHomeRemote
com.titan.cabin.CabinRemote
java.lang.Integer
false
Master Suite
```

Exercise 5.2: The EJBObject, Handle, and Primary Key

In this exercise you will compile three additional client programs and test them against the TravelAgent and Cabin beans.

Step 1: Compile the classes

Open a Windows command prompt and `cd` to this directory:

`<InstallDrive>/EJBWorkbook/Source/Chapter5/Exercise5-2/dev`

There, run *Exercise5_2Compile.bat* to compile the classes for this exercise. Verify that compilation succeeded and that there are `.class` files for all of the `.java` files in the *Exercise5-2* directory structure before moving on.

Step 2: Create three new application client .jar files

As in Exercise 5.1, you will use the AAT's Create Application Client wizard to create three new client *.jar* files, using the information provided below.

Create an application client for the *Client_53* class, entering:

- ◆ **FileName:** `Client_53.jar`
- ◆ **Display Name:** `Client_53`
- ◆ **Files to import:** `com.titan.clients.Client_53.class` (browse the *Exercise5-2/dev* directory)
- ◆ **Main Class:** `com.titan.clients.Client_53.class`
- ◆ No icons, EJB references, resource references, or environment entries

Save the file as *Exercise5-2/dev/Client_53.jar*. Next, create a *.jar* for the *Client_54* class:

- ◆ **FileName:** `Client_54.jar`
- ◆ **Display Name:** `Client_54`
- ◆ **Files to import:** `com.titan.clients.Client_54.class` (browse *Exercise5-2/dev*)
- ◆ **Main Class:** `com.titan.clients.Client_54.class`
- ◆ No icons, EJB references, resource references, or environment entries

Save the file as *Exercise5-2/dev/Client_54.jar*. Finally, run the wizard a third time and enter:

- ◆ **FileName:** `Client_55.jar`

- ◆ **Display Name:** `Client_55`
- ◆ **Files to import:** `com.titan.clients.Client_55.class` (browse *Exercise5-2/dev*)
- ◆ **Main Class:** `com.titan.clients.Client_55.class`
- ◆ No icons, EJB references, resource references, or environment entries

Save the file as *Exercise5-2/dev/Client_55.jar* and move on.

Step 3: Create the Client_53.ear file

Using the AAT, create an *.ear* containing the three new application clients, *travelagent.jar*, and *cabin.jar*, and save it as *Exercise5-2/dev/Client_53.ear*. Finally, generate the deployed code as before.

Step 4: Test the application clients

Ensure that the application server is running and that *Exercise5-2/dev* is the current directory, then type:

```
launchClient Deployed_Client_53.ear -CCjar=Client_53.jar
```

Verify that the output you see on the console matches the following:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.clients.Client_53
com.titan.travelagent.TravelAgentHomeRemote
com.titan.travelagent.TravelAgentRemote
true
```

Next, execute:

```
launchClient Deployed_Client_53.ear -CCjar=Client_54.jar
```

Verify the output against:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
```

```
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.clients.Client_54
Creating Cabin 101 and retrieving additional reference by pk
Testing reference equivalence
Names match!
cabin_1.isIdentical(cabin_2) returns true - This is correct
Testing serialization of primary key
Setting cabin name to Presidential Suite
Writing primary key object to file...
Reading primary key object from file...
Acquiring reference using deserialized primary key...
Retrieving name of Cabin using new remote reference...
Presidential Suite
Removing Cabin from database to clean up..
```

Finally, run *Client_55*:

```
launchClient Deployed_Client_53.ear -CCjar=Client_55.jar
```

And verify its output against:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.clients.Client_55
Testing serialization of EJBObjct handle
Writing handle to file...
Reading handle from file...
Acquiring reference using deserialized handle...
cabin_1.isIdentical(cabin_2) returns true - This is correct
Testing serialization of Home handle
Writing Home handle to file...
Reading Home handle from file...
Acquiring reference using deserialized Home handle...
Acquiring reference to bean using new Home interface...
cabin_1.isIdentical(cabin_3) returns true - This is correct
```


Exercise for Chapter 9

Exercise 9.1: A CMP 1.1 Entity Bean

In this exercise you will compile and deploy the Ship CMP entity bean. You will also compile and deploy two client applications, named *ShipTestClient* and *ShipTestClientUndo*, as J2EE application clients to test the Ship bean. To accomplish these tasks, you will:

1. Learn how custom finder methods are implemented in WebSphere Application Server AEs.
2. Compile the Java source code for the EJB and the client programs into *.class* files
3. Create an EJB *.jar* file for the Ship bean
4. Use the AAT to assemble Java application client *.jar* files for *ShipTestClient* and *ShipTestClientUndo*.
5. Use the AAT to assemble the EJB *.jar* file and the two application client *.jar* files into an enterprise archive (*.ear*) file and use the AAT to generate deployment code for the *.ear* file
6. Use the DB2 Command Line Processor (CLP) to create a table for the Ship bean in the DB2 [SAMPLE](#) database.
7. Use the Administrative Console to deploy the new *.ear* file
8. Test your client applications with the *launchClient* tool in WebSphere Application Server AEs

Step 1: Learn about WebSphere custom finder methods

If you examine the files in the */Chapter9/dev/META-INF* directory you will find a file that was not described in Chapter 9 of the EJB book, named *ibm-ejb-jar-ext.xmi*. Open it:

```
<ejbext:EJBJarExtension xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns:ejbext="ejbext.xmi"
xmlns:ejb="ejb.xmi" xmi:id="ejb-jar_ID_Ext">
  <ejbJar href="META-INF/ejb-jar.xml#ejb-jar_ID"/>
  <ejbExtensions xmi:type="ejbext:ContainerManagedEntityExtension"
xmi:id="ContainerManagedEntityExtension_1">
    <enterpriseBean xmi:type="ejb:ContainerManagedEntity"
href="META-INF/ejb-jar.xml#ContainerManagedEntity_1"/>
    <structure xmi:id="BeanStructure_1" inheritanceRoot="false"/>
    <beanCache xmi:id="BeanCache_1" activateAt="TRANSACTION"
loadAt="TRANSACTION"/>
    <internationalization xmi:id="BeanInternationalization_1"
invocationLocale="CALLER"/>
    <localTran xmi:id="LocalTran_1" boundary="BEAN_METHOD"
unresolvedAction="ROLLBACK"/>
    <finderDescriptors xmi:type="ejbext:WhereClauseFinderDescriptor"
xmi:id="WhereClauseFinderDescriptor_1" whereClause="T1.CAPACITY=?">
```

```
<finderMethodElements xmi:id="MethodElement_1"
name="findByCapacity" parms="int" type="Home">
  <enterpriseBean xmi:type="ejb:ContainerManagedEntity"
href="META-INF/ejb-jar.xml#ContainerManagedEntity_1"/>
</finderMethodElements>
</finderDescriptors>
</ejbExtensions>
</ejbext:EJBJarExtension>
```

You may wonder what the purpose of this XML file is. This file is unique to WebSphere and is called a *deployment descriptor extension document*. To understand it, recall that the EJB book mentions that the EJB 1.1 specification does not specify how a finder method must work; the implementation is left to the vendor's discretion. In fact, WebSphere supports four ways to implement a custom finder method. Each way corresponds to a special `xsi:type` parameter in the `<finderDescriptors>` tag shown above. They are:

- ◆ The SQL `SELECT` custom finder

You can provide an entire SQL `SELECT` statement in an XML tag of the *ibm-ejb-jar-ext.xml* file. This particular implementation choice has been deprecated, and is included only for compatibility with earlier releases of WebSphere. This corresponds to the `xsi:type:ejbext:FullSelectFinderDescriptor`.

- ◆ The SQL `WHERE` custom finder (shown above)

You can provide only those parts of an SQL statement that would follow a `WHERE` in a SQL `SELECT` statement. You may provide query parameters (`?` characters) to the `WHERE` clause, and these will be substituted by the arguments to the corresponding finder method at run time. This corresponds to the `ejbext:WhereClauseFinderDescriptor` shown above.

- ◆ The Method custom finder

You can provide a method name in this tag that corresponds to a method in a class you provide named `beanClassNameFinderObject` (where `beanClassName` is the name of your bean implementation class). The method should return a `java.sql.PreparedStatement` that can be executed to obtain the rows for the custom finder. This corresponds to the `xsi:type` parameter `ejbext:UserFinderDescriptor`.

- ◆ The EJB Query Language custom finder

You can provide a statement in a subset of the EJB 2.0 EJB Query Language that will allow you to select EJBs based on the `cmp-field` attributes of this bean type. This corresponds to the `xsi:type` parameter `ejbext:EjbqlFinderDescriptor`

- ◆ Note that this capability is a technology preview in WebSphere 4.0 and should not yet be considered for use in production applications.

For each of these choices, you will provide a `<finderDescriptors>` tag with the appropriate `xsi:type` choice and the required parameters for that choice, and a `<finderMethodElements>` tag that identifies the method to which the `finderDescriptor` applies. Each of these is described in greater detail in the product documentation. In particular, you should read the section in the WebSphere AEs InfoCenter on “Implementing custom finders in home interfaces for CMP Entity beans.”

The complexities of writing custom finders using the `Method` approach or the EJB Query Language approach are beyond the scope of this workbook. Because we are using a SQL `WHERE` custom finder, which is by far the most common type, however, we will take this particular example apart to give you a better understanding as to how to write your own custom finders with WebSphere. Let’s begin by looking again at the remote interface for our Ship bean, `ShipHomeRemote`:

```
public interface ShipHomeRemote extends javax.ejb.EJBHome {

    public Enumeration findByCapacity(int capacity)
        throws FinderException, RemoteException;

}
```

Notice that the `findByCapacity()` method described above takes a single `int` parameter, `capacity`, which the method will use to find Ship EJBs of a specified capacity. Now, because we know that each Ship entity bean corresponds one-to-one to a row in a table in our relational database (for which you will see the DDL in Step 6), we can surmise that finding the right EJBs by running our finder method is equivalent to finding the corresponding rows in the relational database that match an equivalent query. In our case the query that would return all of the rows in the `ShipEJB` table that match a particular capacity would be:

```
Select * from ShipEJB T1 where T1.capacity = ?
```

Note the single question mark in this query. In JDBC `PreparedStatement`s, question marks are placeholders that are replaced by actual parameters at run time. The corresponding JDBC code WebSphere would generate to return the rows that match our query could look like the following (but more complex; we have simplified the code for learning purposes):

```
/**
 * Return a ResultSet that contains a set of rows from the database
 * where the capacity is equal to the supplied capacity.
 */

java.sql.ResultSet getResultsForCapacity(int capacity) {
    // get a connection first (not shown for brevity)
    String query = "select * from ShipEJB T1 where T1.capacity = ?";
    PreparedStatement pStmt = connection.prepareStatement(query);
    pStmt.setInt(1, capacity);
    ResultSet results = pStmt.executeQuery();
    return results;
}
```

The key thing to note is that these substituted parameters are positional. Because `capacity` is the only parameter, it could be substituted for only one choice in the SQL statement: the first question mark. A slightly more complex version of the above shows how position becomes important in larger queries:

```
/**
 * Return a ResultSet that contains a set of rows from the database
 * where the capacity is equal to the supplied capacity and the
 * name is equal to the supplied name.
 */

java.sql.ResultSet getResultsForCapacityAndName(
    int capacity, String name) {
    // get a connection first (not shown for brevity)
    String query = "select * from ShipEJB T1 where" +
        " T1.capacity = ? AND T1.name = ?";
    PreparedStatement pstmt = connection.prepareStatement(query);
    pstmt.setInt(1, capacity);
    pstmt.setString(2, name);
    ResultSet results = pstmt.executeQuery();
    return results;
}
```

Note that here we are providing two parameters to this method; a capacity and a name. The SQL `select` statement shown in this method includes two question marks. The code that follows the `prepareStatement()` method substitutes these actual parameters into the SQL in the same order in which the parameters were provided.

This kind of matching of method parameters to the order of SQL substitution parameters is exactly what WebSphere will do for you automatically in an SQL Where custom finder. In fact, WebSphere will generate code in its deployment classes nearly identical to the above. The relevant lines from the `ibm-ejb-jar-ext.xmi` file that make up the `Where` custom finder are shown below:

```
<finderDescriptors xmi:type="ejbext:WhereClauseFinderDescriptor"
xmi:id="WhereClauseFinderDescriptor_1" whereClause="T1.CAPACITY=?">
    <finderMethodElements xmi:id="MethodElement_1"
name="findByCapacity" parms="int" type="Home">
        <enterpriseBean xmi:type="ejb:ContainerManagedEntity"
href="META-INF/ejb-jar.xml#ContainerManagedEntity_1"/>
    </finderMethodElements>
</finderDescriptors>
```

Taking these lines apart one by one will help you understand how this works. Begin by looking at the first line:

```
<finderDescriptors xmi:type="ejbext:WhereClauseFinderDescriptor"
xmi:id="WhereClauseFinderDescriptor_1" whereClause="T1.CAPACITY=?">
```

This defines the type of finder we are building, `ejbext:WhereClauseFinderDescriptor`, and provides both an id for this `Finder` descriptor (which must be unique within this file) and the `WHERE` clause that WebSphere will use to build the SQL needed to retrieve the rows that match the EJBs you wish to find. Now look at the second line:

```
||| <finderMethodElements xmi:id="MethodElement_1" name="findByCapacity"
|||   parms="int" type="Home">
```

This line defines the other elements of the custom finder method – namely which method this `CustomFinder` is defined for (`name=findByCapacity`), and the list of parameter types that the method takes (`parms=int`). Each of these parameter types is specified in the order in which its corresponding parameters are specified in the home interface.

Where there is more than one parameter in the EJB custom finder method, as in a method like `findByCapacityAndName(int capacity, String name)`, you would need to separate the parameter types by spaces:

```
||| <finderMethodElements xmi:id="MethodElement_2"
|||   name="findByCapacityAndName" parms="int java.lang.String"
|||   type="Home">
```

Only one more line in this segment of XML needs explanation. After the opening `<finderMethodElements>` tag, there is another tag:

```
||| <enterpriseBean xmi:type="ejb:ContainerManagedEntity" href="META-
|||   INF/ejb-jar.xml#ContainerManagedEntity_1"/>
```

What this tag does is to link together a particular EJB inside the *ejb-jar* file (in this case the Ship bean) with the finder method being described, through a reference to the ID of an EJB inside the *EJB-JAR.XML* file. At this point, you may be thinking “but there are no IDs inside *EJB-JAR.XML*.” Well, you are right. The XML deployment descriptor supplied in the EJB book does not contain any (optional) ID tags. However, since WebSphere knows that it will need unique IDs to correspond to each EJB, both the AAT and the WebSphere console will insert them into the XML when you first deploy the EJB. For instance, a segment of the updated XML that is inside the fully deployed EJB is shown below:

```
||| <ejb-jar id="ejb-jar_ID">
|||   <enterprise-beans>
|||     <entity id="ContainerManagedEntity_1">
|||       <description>This bean represents a cruise ship.</description>
|||       <ejb-name>ShipEJB</ejb-name>
```

Note that the generated ID, `ContainerManagedEntity_1`, matches the ID that the `<enterpriseBean>` tag refers to above. Now, how do you keep these IDs straight in your own EJB deployment descriptors? You have two choices:

- ♦ You can let the AAT automatically generate them, then patch the `ibm-ejb-jar-ext.xmi` file after the initial generation (which is probably easiest).
- ♦ You can insert the IDs into your deployment descriptors from the start (which is probably safest).

Step 2: Compile the Ship classes

Now that you understand how custom finders work in WebSphere, you are ready to begin working through the example. To compile the `.java` files in the `Chapter9/dev` directory structure, simply open a command prompt and execute the `ShipCompile.bat` batch file. Afterward verify that there are `.class` files for all of the `.java` files in the directory structure rooted at `/dev/com/titan`.

Step 3: Build the Ship EJB .jar file

To build the `ship.jar` EJB `.jar` file, just execute `Chapter9/dev/BuildShipJarFile.bat`. This operation will assemble the compiled classes and the `ejb-jar.xml` file appropriately.

Step 4: Build the application client .jar files

When you were reading Chapter 9 of the EJB book you may have noticed that it's slightly different from the other chapters with exercises, in that it does not contain any sample programs to test the EJBs defined in it. To determine whether your new Ship EJB functions correctly, you will have to develop a sample program to demonstrate its functionality – in particular, the one new feature added to the Ship bean that was not present in any of the CMP EJBs developed in Chapter 4: the custom finder method `findByCapacity`.

This sample, `ShipTestClient`, creates two Ships with different capacities, then calls `findByCapacity` to retrieve only those Ships with a specified capacity. The source code for the sample is shown below. You will want to review it and make sure you understand how it operates:

```
package com.titan.shipclient;

import javax.naming.*;
import com.titan.ship.*;
import java.util.*;

public class ShipTestClient {

    public static void main(String[] args) {
        try {
            Context jndiContext = getInitialContext();
            Object ref = jndiContext.lookup("ShipHomeRemote");
            ShipHomeRemote home =
                (ShipHomeRemote)
                javax.rmi.PortableRemoteObject.narrow(
                    ref,
                    ShipHomeRemote.class);
            ShipRemote ship1 = home.create(
                new Integer(1), "Andrea Doria", 2000, 2500);

            System.out.println("Adding ship 1");
            System.out.println(ship1.getName());
            System.out.println(ship1.getCapacity());
            System.out.println(ship1.getTonnage());

            ShipRemote ship2 = home.create(
                new Integer(2), "Marie Celeste", 5000, 2500);

            System.out.println("Adding ship 2");
            System.out.println(ship2.getName());
            System.out.println(ship2.getCapacity());
            System.out.println(ship2.getTonnage());

            System.out.println("Ships of capacity = 5000 are:");
            Enumeration enum = home.findByCapacity(5000);
            while (enum.hasMoreElements()) {
                Object shipRef = enum.nextElement();
                ShipRemote ship = (ShipRemote)
                    javax.rmi.PortableRemoteObject.narrow(
                        shipRef, ShipRemote.class);
                System.out.println(ship.getName());
            }

        } catch (java.rmi.RemoteException re) {
            re.printStackTrace();
        } catch (javax.naming.NamingException ne) {
```



```
        ne.printStackTrace();
    } catch (javax.ejb.CreateException ce) {
        ce.printStackTrace();
    } catch (javax.ejb.FinderException fe) {
        fe.printStackTrace();
    }
}

public static Context getInitialContext()
    throws javax.naming.NamingException {

    java.util.Properties properties =
        new java.util.Properties();
    properties.put(
        javax.naming.Context.PROVIDER_URL, "iiop:///");
    properties.put(
        javax.naming.Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
    InitialContext initialContext =
        new InitialContext(properties);
    return initialContext;
}
}
```

You don't want to leave the database cluttered with Ships that might distract you in later examples. *ShipTestClientUndo* will remove the newly added Ships from the database. The source code for this program is shown below. Review it before creating the application client *jar* file to be certain you know what it does:

```
package com.titan.shipclient;

import javax.naming.*;
import com.titan.ship.*;

public class ShipTestClientUndo {

    public static void main(String[] args) {
        try {
            Context jndiContext = getInitialContext();
            Object ref = jndiContext.lookup("ShipHomeRemote");
            ShipHomeRemote home =
                (ShipHomeRemote)
                javax.rmi.PortableRemoteObject.narrow(
                    ref,
                    ShipHomeRemote.class);

            ShipRemote ship1 =
                home.findByPrimaryKey(new Integer(1));
            ship1.remove();

            ShipRemote ship2 =
                home.findByPrimaryKey(new Integer(2));
            ship2.remove();

            System.out.println("Undo Completed Successfully");

        } catch (java.rmi.RemoteException re) {
            re.printStackTrace();
        } catch (javax.naming.NamingException ne) {
            ne.printStackTrace();
        } catch (javax.ejb.RemoveException re) {
            re.printStackTrace();
        } catch (javax.ejb.FinderException fe) {
            fe.printStackTrace();
        }
    }
}
```

```
public static Context getInitialContext()
    throws javax.naming.NamingException {

    java.util.Properties properties =
        new java.util.Properties();
    properties.put(
        javax.naming.Context.PROVIDER_URL, "iiop:///");
    properties.put(
        javax.naming.Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
    InitialContext initialContext =
        new InitialContext(properties);
    return initialContext;
}
```

Now that you've seen the source code for the examples, and have some idea what they will do when you run them, you are ready to build application client *.jar* files for them. You've already done this several times, so the procedure should seem familiar. If not, review the procedure in Exercise 4.1 before proceeding with this exercise.

Begin by opening the AAT and selecting the **Create Application Client** wizard. Enter the following information to create your first application client:

- ◆ **File Name:** **ShipTestClient.jar**
- ◆ **Display Name:** **ShipTestClient**
- ◆ **Files to import:** **com.titan.shipclient.ShipTestClient.class** (in the file import dialog select the *Chapter9/dev* directory)
- ◆ **Main Class:** **com.titan.shipclient.ShipTestClient.class**
- ◆ No icons, EJB references, resource references, or environment entries

When you have finished the wizard, save this client as *Chapter9/dev/ShipTestClient.jar*. Next, open the wizard again and enter the following to create the second application client:

- ◆ **FileName:** **ShipTestClientUndo.jar**
- ◆ **Display Name:** **ShipTestClientUndo**
- ◆ **Files to import:** **com.titan.shipclient.ShipTestClientUndo.class** (in the *Chapter9/dev* directory)
- ◆ **Main Class:** **com.titan.shipclient.ShipTestClientUndo.class**
- ◆ No icons, EJB references, resource references, or environment entries

When you finish the wizard, save this client *.jar* file as *Chapter9/dev/ShipTestClientUndo.jar*.

Step 5: Build the Ship .ear file

Using the AAT, assemble the two application clients created in Step 3 and the *ship.jar* EJB *.jar* file created in Step 2 to create an *.ear* file. When you are finished, save this new *.ear* file as *Chapter9/dev/Ship.ear*. Finally, use the AAT's **Generate Deployed Code** menu option to generate the deployed *.ear* file, as in earlier exercises.

Step 6: Create the ShipEJB table

Because you are creating a new CMP EJB, you will need to create a table in DB2 for your EJB. If you are using a different database, remember you will need to examine the contents of the *table.ddl* file in the deployed *.ear* file, and use your database's tools to enter the SQL found there.

If you are using DB2, follow the steps outlined earlier to open a DB2 command-line processor and enter the following SQL commands to create the table and set its primary key:

```
CREATE TABLE SHIPEJB
  (ID INTEGER NOT NULL,
   NAME VARCHAR(250) ,
   CAPACITY INTEGER,
   TONNAGE DOUBLE)

ALTER TABLE SHIPEJB
  ADD CONSTRAINT SHIPEJBPK PRIMARY KEY ("ID")
```

Step 7: Deploy the EJB .jar file for the Ship bean

Follow the directions previously outlined in Exercise 4.1, Step 8, to deploy your new *Ship.jar* to IBM WebSphere Application Server AEs. Remember to start the application server before opening a browser on the web console. Also be careful to:

1. Deploy the EJB with the JNDI name **ShipHomeRemote** (the name used in the source for this exercise's clients).
2. Save the configuration
3. Stop and restart the server

Step 8: Run the Ship test clients

Now you are ready to run the test client programs to test your `ShipEJB`. Open a command prompt, `cd` to the *Chapter9/dev* directory, and enter the following command:

```
launchClient Deployed_Ship.ear -CCjar=ShipTestClient.jar
```

Verify the client's output against the following:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001

WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.shipclient.ShipTestClient
Adding ship 1
Andrea Doria
2000
2500.0
Adding ship 2
Marie Celeste
5000
2500.0
Ships of capacity = 5000 are:
Marie Celeste
```

Now clean up the database by running the Undo Test client.

```
launchClient Deployed_Ship.ear -CCjar=ShipTestClientUndo.jar
```

Verify that you receive the following output, with no error messages or stack traces:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001

WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.shipclient.ShipTestClientUndo
Undo Completed Successfully
```


Exercise for Chapter 10

Exercise 10.1: A BMP Entity Bean

In this exercise you will compile and deploy a BMP version of the Ship entity bean. You will also compile and deploy two J2EE client applications to test the Ship BMP bean.

Step 1: Compile the Source Files

To compile the class files for this exercise, open a command prompt, change directories to *Chapter10/dev*, and execute *ShipCompile.bat*. Afterward verify that there are *.class* files for all of the *.java* files in the directory structure rooted at *dev/com/titan*.

Step 2: Assemble the EJB .jar file

To assemble the compiled classes and the *ejb-jar.xml* file appropriately and build the *bmpship.jar* EJB *jar* file, execute *BuildShipJarFile.bat* in the *Chapter10/dev* directory.

Step 3: Create the application client .jar files

You are now ready to build the application client *.jar* files for these two example programs, just as you have done in all previous exercises.

Begin by opening the AAT and selecting the **Create Application Client** wizard. Enter the following information to create your first application client:

- ◆ **File Name:** **ShipTestClient.jar**
- ◆ **Display Name:** **ShipTestClient**
- ◆ **Files to import:** **com.titan.shipclient.ShipTestClient.class** (in the file import dialog select the *Chapter10/dev* directory)
- ◆ **Main Class:** **com.titan.shipclient.ShipTestClient.class**
- ◆ No icons, EJB references, resource references, or environment entries

After completing the wizard, save the new client *.jar* file to *Chapter10/dev/ShipTestClient.jar*. Then repeat the steps to create a second *.jar* file with the following attributes:

- ◆ **File Name:** **ShipTestClientUndo.jar**
- ◆ **Display Name:** **ShipTestClientUndo**
- ◆ **Files to import:** **com.titan.shipclient.ShipTestClientUndo.class** (select *Chapter10/dev*)
- ◆ **Main Class:** **com.titan.shipclient.ShipTestClientUndo.class**

- ◆ No icons, EJB references, resource references, or environment entries

Complete this step by saving the second `.jar` file as *Chapter10/dev/ShipTestClientUndo.jar*.

Step 4: Assemble the .ear file and generate the deployed .ear file

Using the AAT, assemble the two application clients created in Step 3 and the *bmpship.jar* EJB *.jar* file created in Step 2 to create an *.ear* file. When you are finished, save this new *.ear* file as *Chapter10/dev/bmpship.ear*. Finally, use the AAT's **Generate Deployed Code** menu option to generate the deployed *.ear* file, as earlier.

Step 5: Create the Ship table

One of the drawbacks of bean-managed persistence is that, because the SQL is embedded in the Java code, it is not as resilient in the face of changes in the database structure. When you use container-managed persistence, a database change obliges you only to change the mapping metadata and redeploy the affected beans.

You can see an example of this difference in the BMP bean described in the EJB book. The BMP version of the Ship bean executes SQL statements that imply a particular table structure. To use this bean, therefore, you will need to create an appropriately structured `SHIP` table in DB2, using the SQL `CREATE` statement found in Chapter 9 of the EJB book.

To create the table and set its primary key, open a DB2 command-line processor and enter the following SQL command:

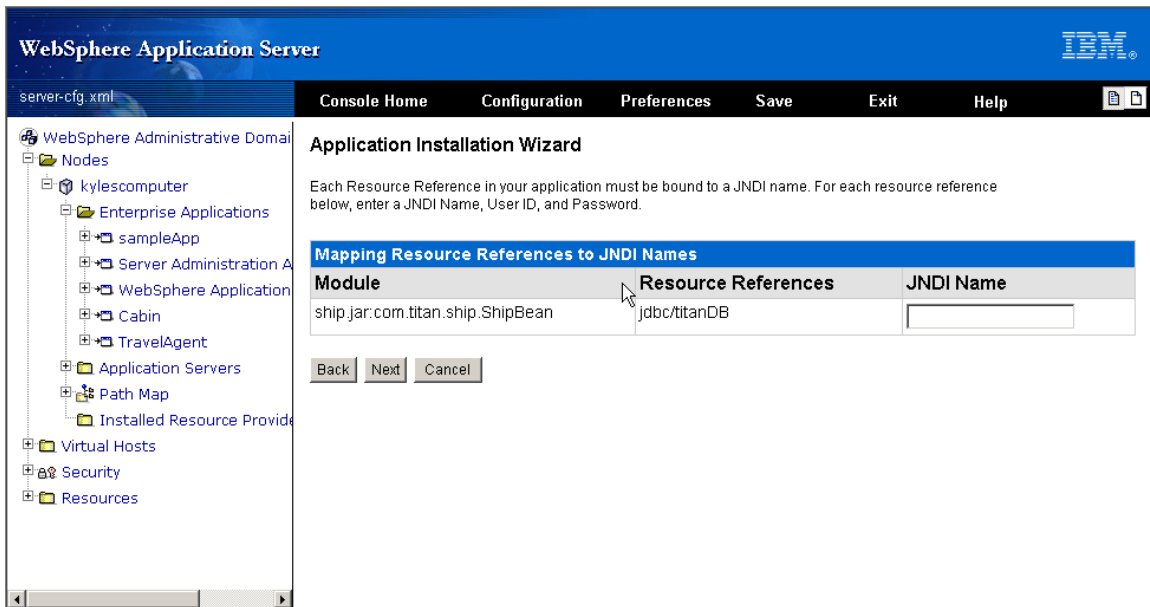
```
CREATE TABLE SHIP (ID INT PRIMARY KEY NOT NULL, NAME CHAR(30),  
CAPACITY INT, TONNAGE DECIMAL(8,2))
```

Step 6: Deploy the EJB .jar file into WebSphere AEs

You have already deployed three EJBs into IBM WebSphere Application Server AEs, so the procedure should be familiar to you by now. If not, review the instructions in exercise 4.1.

In this exercise, you will deploy the *bmpship.jar* EJB file. In that process you will need to choose a name for the enterprise application; you will want to make sure you choose one that has not yet been used. We suggest you use the application name *BMPShip*. Also, you will want to deploy the EJB with the JNDI name used by the client applications, *BmpShipHomeRemote*. Otherwise, proceed through the deployment wizard as usual. After you set the EJB's JNDI name, you will come to a screen you have not yet encountered in WebSphere deployment:

Figure 77: Setting resource references



You see this page because the deployment descriptor for this bean includes a resource reference – it declares a data source with the name `jdbc/titanDB`. You need to map this internal reference to a data source in the global JNDI namespace. You have already created the `jdbc/SampleDB` data source, which points to the `SAMPLE` database in DB2, for earlier exercises, and it will work fine for this one. Enter `jdbc/SampleDB` in the text field on this page and press the **Next** button. At this point you will be able to confirm your settings and finish deploying the EJB.

As always, after you finish deployment, remember to save the configuration, and to stop and restart the server.

Step 7: Test the new EJB with the test clients

Change to the `Chapter10/dev` directory and launch the test client with

```
launchClient Deployed_bmpship.ear -CCjar=ShipTestClient.jar
```

Its output should be exactly the same as you saw in Exercise 9.1. After you have verified that the output is correct, run the `Undo` client:

```
launchClient Deployed_bmpship.ear -CCjar=ShipTestClientUndo.jar
```

Complete the exercise by verifying that the `undo` has completed successfully.

Exercises for Chapter 12

Exercise 12.1: A Stateless Session Bean

In this exercise you will develop the `ProcessPayment` EJB, together with a set of other EJBs used in payment processing. This exercise represents the first time that you will encounter a problem whose complexity approaches the level you will see in your own applications. You will need to pay close attention to the best practices and procedures outlined in this chapter.

You may want to begin by opening the `Chapter12/Exercise12-1/dev/META-INF/ejb-jar.xml` file and scanning its contents. You will find that it differs from all of the other EJB deployment descriptor files you have seen in the EJB book and in this workbook, in that it contains, not just one, but several EJBs. In fact, deployment of a single EJB is not common practice. Most EJBs do not work alone, but instead work as parts of a larger group of EJBs that are related by a common purpose. In everyday practice, rather than deploy each EJB individually, you will deploy them in groups, as you will in this example. What's more, EJB `.jar` files are not normally deployed alone, but in sets that compose a larger enterprise application. As you progress through this exercise, you will see that this larger scope changes the deployment process.

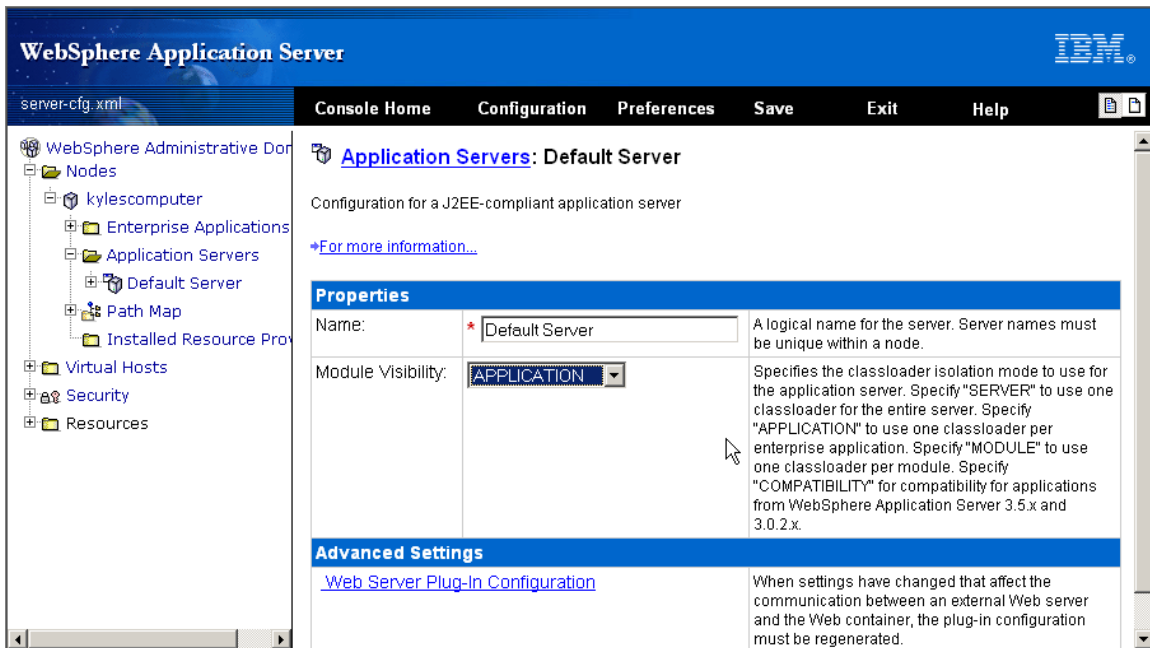
Two of the fundamental problems in building J2EE applications are determining how multiple EJB `.jar` files should be packaged into `.ear` files, and determining how classes will find the other classes they need at run time. We will address both of these issues in this exercise.

You may recall that in Exercise 4.2 we encountered the problem of needing to “bring in” parts of the `Cabin` EJB to the `TravelAgent` `.jar` file in order to make the `TravelAgent` EJB work. We pointed out that this approach is not a “best practice” because the two copies of the code (one in `Cabin.jar` and another in `TravelAgent.jar`) may diverge over time. The better solution is to allow the EJBs in one `.jar` file to see the classes in the other somehow. You can accomplish this result by making one `.jar` file part of the classpath of the other. There are two basic ways to accomplish this:

- ◆ You can change the classpath visibility settings of the application server as a whole
- ◆ You can change the manifest files in the EJB `.jar` files to allow one EJB `.jar` to include another EJB `.jar` in its classpath.

Begin by examining the first option. WebSphere AEs has four “modes” of classpath visibility, which the Administrative Console refers to as *module visibility*. You specify visibility for each instance of the application server, which you can reach by expanding your node, expanding the Application Servers tab, and then selecting the Default Server tab, as in the following figure:

Figure 78: Setting module visibility



The four modes are:

- ◆ **Compatibility** – This mode is designed for compatibility with the 3.x versions of WebSphere. In Compatibility mode, every EJB *.jar* module is included in every other's classpath. If you deploy an EJB that references another EJB you do not need to update the *.jar* files in any way to allow them to find and load the correct Remote and Home interfaces and RMI stub classes. Likewise, *.war* modules are visible to all EJB *.jar* modules; *.war* modules are **not** visible to other *.war* modules, however.
- ◆ **Application** – In this mode, all *.wars* and EJB modules in a single *.ear* file are visible to each other, but not to modules outside that *.ear* file.
- ◆ **Module** – In this mode, every *.war* and EJB module is separate, and they are not visible to each other.
- ◆ **Server** – In this mode, all *.war* and EJB modules are visible to all other modules, regardless of the application (*.ear*) they were deployed into.

The default mode for WebSphere AEs is Application visibility, which is also the recommended visibility for most server configurations. IBM chose this default to maintain compatibility with the upcoming J2EE 1.3 specification, which makes this level of visibility standard.

The second way in which an EJB can reference classes it needs that are outside its own *.jar* file is to extend the classpath of an EJB *.jar* file by adding classpath lines to the *manifest file* of the *.jar*. The manifest file is a descriptor file specific to each *.jar* file that can contain information to be used by the class loader that loads the classes in the *.jar* file. The manifest file is always named *Manifest.mf*, and is always located in the *META-INF* directory off the root of the *.jar* file.

For example, in this exercise you will need to build a *.jar* file named *processpayment.jar*, which contains the Reservation entity bean from the EJB book. That EJB needs to be able to use the CabinEntity EJB that we defined in Chapter 4, which was placed in its own *.jar* file, *cabin.jar*. So, how do you allow the *processpayment.jar* file to refer to the *cabin.jar* file? Look at these two lines, from *META-INF/Manifest.mf*:

```
Manifest-Version: 1.0
Class-Path:  cabin.jar
```

The first line is required, and simply specifies the version number of this manifest file. You're interesting in the second line, which specifies any additional *.jar* files that should be added to this *.jar* file's classpath. If you need to specify multiple *.jar* files here, you simply separate their names by spaces. All entries in this line are specified relative to the root of the *.jar* file, so the class loader will expect *cabin.jar* to be located at the root of the *.jar* file.

If WebSphere already resolves classpath issues like this one when you use the Application Module Visibility mode, why do you need to care about visibility? The reason is that there are two times when classes in a *.jar* file need to refer to classes in another *.jar* file – run time (which is what Module Visibility refers to) and *compile* time. The primary concern is resolving compile-time references. When WebSphere generates *.java* files for the RMI-IIOP stubs in all EJBs, and when it generates persistence classes for entity beans, it also compiles those classes and inserts the resulting *.class* files back into the deployed *.jar* file for use by the application server. The compiler will allow you to include a reference to a file only if that file is found on the compilation classpath. The *Class-Path* entry in your EJB *.jar* files provides information, not only for the run-time classpath, but for the compile-time classpath as well.

That Application visibility is the standard set by the J2EE 1.3 specification leads you to a cardinal rule of J2EE packaging: to ensure portability across application servers all *.ear* files should be self-sufficient (that is, they should contain the entire graph of all referenced classes not provided by the J2EE libraries). This is a good rule for a number of other reasons too, however. WebSphere allows classes to be inserted in a number of other places in the run-time classpath as well; the InfoCenter describes a number of these locations in detail.

These locations have a common attribute, though; they are shared across all of the enterprise applications deployed on an application server, with consequences that may not be immediately obvious. For instance, if you tried to place the *cabin.jar* file on the System classpath (which is set in the JVM Settings tab of the Administrative Console) you would find out that the same *.jar* file was now visible to all enterprise applications in that server. This “global scope” would be a problem if any application needed to use a different version of the *cabin.jar* file than the one in the System classpath. The best way to avoid these kinds of version-dependency issues is simply to make each *.ear* entirely self-contained.

To be self-contained in that way, our new *processpayment.ear* file must contain:

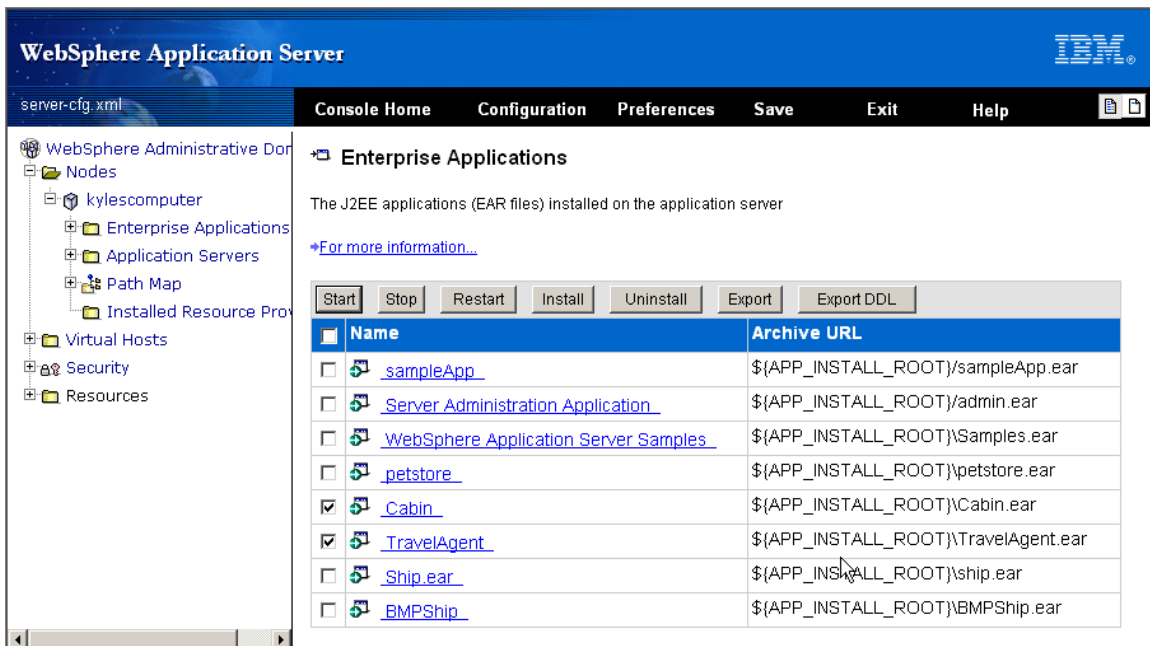
- ◆ A *processpayment.jar* file that contains the new classes defined in Chapter 12
- ◆ The *cabin.jar* file from Exercise 4.1
- ◆ The *travelagent.jar* file from Exercise 4.2

To deploy this new *.ear* file, you will have to change the configuration of your application server. You will find that all of the previous examples will still run, however, because you will not change any of the JNDI names of the EJBs. You will simply change the containment structure for the EJBs by grouping them in a single *.ear* file.

Step 1: Undeploy the existing Cabin and TravelAgent beans

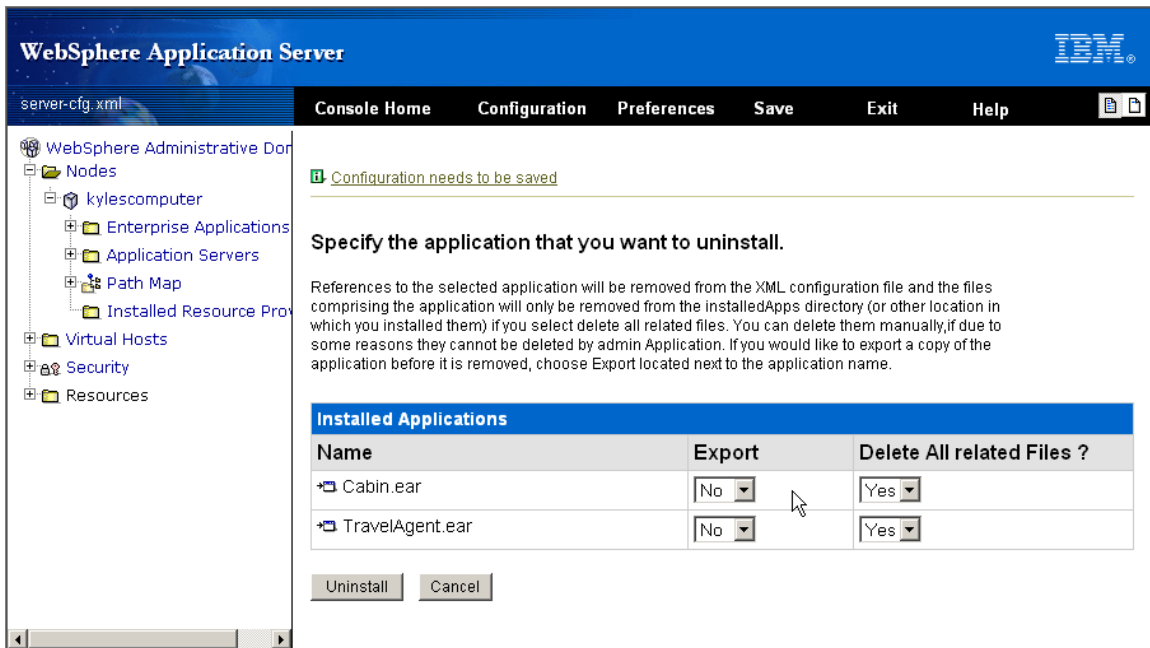
Begin the process of changing your application server configuration by ensuring that the application server has started. Open a WebSphere Administrative Console. Expand your node and select the **Enterprise Applications** tab. Select both the *TravelAgent* and *Cabin* applications from the list and press the **Stop** Button.

Figure 79: Selecting applications to be uninstalled



WebSphere will then stop those two applications, which will change the icon next to the application names. Next, select the two applications a second time and press the **Uninstall** button. You will see the following confirmation page:

Figure 80: Confirming applications to be uninstalled



Be certain you have selected the option to delete the files (set the drop-down to **Yes**), and that you have not chosen to export the files, then press the **Uninstall** button to continue. Once the uninstall is finished, save the new configuration and shut down the application server. When the application server has completely shut down, verify that the *TravelAgent.ear* and *Cabin.ear* directories have been successfully removed from the following directory:

`<WebSphereInstallRoot>/WebSphere/AppServer/InstalledApps`

Step 2: Compile the Source Files

To compile the class files for this exercise, open a command prompt, change to the directory *Chapter12/Exercise12-1/dev*, and execute the *ProcessPaymentBeanCompile.bat* batch file. Afterward verify that there are *.class* files for all of the *.java* files in the directory structure rooted at *dev/com/titan*.

Open the *ProcessPaymentBeanCompile.bat* file in a text editor and you will find an interesting feature:

```
SET JAVA_HOME=%WAS_HOME%\java
SET WS_CLASSPATH=%WAS_HOME%\lib\j2ee.jar

%JAVA_HOME%\bin\javac -classpath
%WS_CLASSPATH%;..\..\..\Chapter4\dev;.
com\titan\processpayment\*.java com\titan\paymentclient\*.java
com\titan\customer\*.java com\titan\reservation\*.java
com\titan\cruise\*.java
```

As you can see, the `CLASSPATH` used by *javac* includes not only the directory structure for this exercise, but the directory for Chapter 4 as well. It must, for the reasons just discussed. As you can see from reading the code in the EJB book, the EJBs in this *jar* file depend on a number of other EJBs, including the Cabin EJB developed in Chapter 4.

Step 3: Assemble the EJB .jar file

To build the *processpayment.jar* EJB *jar* file, in the *Chapter12/Exercise12-1/dev* directory execute *BuildProcessPaymentBeanJarFile.bat*. This batch file will assemble the compiled classes, the *manifest.mf* file, and the *ejb-jar.xml* file appropriately.

Step 4: Create an application client .jar file

As in some previous exercises, you will have to build an example program to demonstrate how the ProcessPayment EJB functions, because the EJB book does not provide one. The following program, which you can find in *Chapter12/Exercise12-1/dev/com/titan/paymentclient*, will test some of the features of the ProcessPayment EJB and will help you determine whether it has been correctly deployed:

```
package com.titan.paymentclient;

import com.titan.customer.*;

import com.titan.processpayment.ProcessPaymentHomeRemote;
import com.titan.processpayment.ProcessPaymentRemote;

import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import java.rmi.RemoteException;
import java.util.Properties;

public class TestPaymentClient {
    public static void main(String [] args) {
        TestPaymentClient client = new TestPaymentClient();
        client.createSampleCustomer();
        client.payByCash();
    }

    public static Context getInitialContext()
        throws javax.naming.NamingException {

        java.util.Properties properties =
            new java.util.Properties();
        properties.put(
            javax.naming.Context.PROVIDER_URL, "iiop:///");
        properties.put(
            javax.naming.Context.INITIAL_CONTEXT_FACTORY,
            "com.ibm.websphere.naming.WsnInitialContextFactory");
        InitialContext initialContext =
            new InitialContext(properties);
        return initialContext;
    }

    public ProcessPaymentHomeRemote getProcessPaymentHome() {
        ProcessPaymentHomeRemote home = null;
        try {
            Context jndiContext = getInitialContext();
            Object ref =
jndiContext.lookup("ProcessPaymentHomeRemote");
            home =
                (ProcessPaymentHomeRemote)
                javax.rmi.PortableRemoteObject.narrow(
                    ref, ProcessPaymentHomeRemote.class);
        } catch (NamingException ne) {
```

```
        ne.printStackTrace();
    }
    return home;
}

public CustomerHomeRemote getCustomerHome() {
    CustomerHomeRemote home = null;
    try {
        Context jndiContext = getInitialContext();
        Object ref = jndiContext.lookup("CustomerHomeRemote");
        home =
            (CustomerHomeRemote)
            javax.rmi.PortableRemoteObject.narrow(
                ref, CustomerHomeRemote.class);
    } catch (NamingException ne) {
        ne.printStackTrace();
    }
    return home;
}

public void createSampleCustomer() {
    try {
        CustomerRemote customer =
            getCustomerHome().create(1, "Kyle", "G", "Brown");
    } catch (java.rmi.RemoteException re) {
        re.printStackTrace();
    } catch (javax.ejb.CreateException ce) {
        ce.printStackTrace();
    }
}

public void payByCash() {
    try {
        Integer pk = new Integer(1);
        CustomerRemote customer =
            getCustomerHome().findByPrimaryKey(pk);
        ProcessPaymentRemote paymentBean =
            getProcessPaymentHome().create();
        paymentBean.byCash(customer, 1000.0);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
```

This program tests only some of the more rudimentary aspects of the *ProcessPayment* bean. It does so by creating a sample Customer EJB, creating a ProcessPayment EJB, and sending it the *byCash* message. This sequence will at least demonstrate that all of the EJBs have been successfully deployed and can be created, and that at least one of the common code paths works correctly. This test client does not include any facilities for verifying that the payment was correctly entered into the database, or for cleaning up the *CUSTOMER* table when it's done. Extending this example to provide these features would be simple, and an excellent exercise for you to try after you have completed this one.

You are now ready to build the application client *jar* files for these two example programs, using the AAT wizard just as in all the previous exercises. Begin by opening the AAT and selecting the **Create Application Client** wizard. Enter the following information to create your application client:

- ◆ **File Name:** `TestPaymentClient.jar`
- ◆ **Display Name:** `TestPaymentClient`
- ◆ **Files to import:** `com.titan.paymentclient.TestPaymentClient.class` (in the file import dialog select the *Chapter12/Exercise12-1/dev* directory)
- ◆ **Main Class:** `com.titan.paymentclient.TestPaymentClient.class`
- ◆ No icons, EJB references, resource references, or environment entries

When you have finished the wizard, save your new application client *jar* file in the *Chapter12/Exercise12-1/dev* directory as *TestPaymentClient.jar*.

Step 5: Assemble the .ear file and generate the deployed .ear file

Using the AAT, create a new *.ear* file that will assemble all of the following:

- ◆ The application client you just created, *TestPaymentClient.jar*
- ◆ The *processpayment.jar* you created in Step 3
- ◆ The *cabin.jar* file from Exercise 4.1
- ◆ The *travelagent.jar* file from Exercise 4.2

Save the new *.ear* file as *Chapter12/Exercise12-1/dev/processpayment.ear*. As in the previous exercises, generate the deployed *.ear* file by using the **Generate Deployed Code** menu option.

Step 6: Create the database tables

In this exercise you will deploy a number of CMP EJBs: Reservation, Cruise, and Customer. You will also need to create the *PAYMENT* table that the ProcessPayment EJB uses, as described in the EJB book.

Open a DB2 command-line processor (CLP) and enter the following SQL command to create the `PAYMENT` table and set its primary key:

```
CREATE TABLE PAYMENT
(customer_id INTEGER,
 amount DECIMAL(8,2),
 type CHAR(10),
 check_bar_code CHAR(50),
 check_number INTEGER,
 credit_number CHAR(20),
 credit_exp_date DATE)
```

Now you need to create the tables for the Reservation, Cruise, and Customer EJBs. One way would be to execute the following SQL statements in the CLP:

```
CREATE TABLE RESERVATIONBEAN
(CUSTOMERID INTEGER,
 CRUISEID INTEGER NOT NULL,
 CABINID INTEGER NOT NULL,
 PRICE DOUBLE,
 DATE DATE)

ALTER TABLE RESERVATIONBEAN
ADD CONSTRAINT RESERVATIONBEANPK PRIMARY KEY (CRUISEID, CABINID)

CREATE TABLE CRUISEBEAN
(ID INTEGER NOT NULL,
 NAME VARCHAR(250),
 SHIPID INTEGER)

ALTER TABLE CRUISEBEAN
ADD CONSTRAINT CRUISEBEANPK PRIMARY KEY (ID)

CREATE TABLE CUSTOMERBEAN
(ID INTEGER NOT NULL,
 LASTNAME VARCHAR(250),
 FIRSTNAME VARCHAR(250),
 MIDDLENAME VARCHAR(250))

ALTER TABLE CUSTOMERBEAN
ADD CONSTRAINT CUSTOMERBEANPK PRIMARY KEY (ID)
```

Because this is a lot of typing, however, you may prefer an alternative route to the same goal. Remember from the earlier discussion about how WebSphere does top-down mapping that, when you generate the deployed code for an EJB *jar* file in the AAT, the AAT generates a file called *Table.ddl* that contains the SQL for creating the tables used by the CMP EJBs.

Table.ddl is stored in the *META-INF* directory of the deployed version of your EJB *.jar* file. If you choose, you can take advantage of it to create your tables without doing a lot of typing:

1. Use an unzip utility or **jar -xvf** to unpack the deployed *.ear* file you created above into a temporary directory.
2. Unpack the deployed *processpayment.jar* file in that temporary directory into another temporary directory.
3. Open a DB2 **Command Window** (not a CLP – this is a different tool, accessed from the IBM DB2 **Start Menu**).
4. Change to the temporary directory where you unpacked *processpayment.jar*
5. Connect to the database by issuing the following command at the prompt:
`db2 connect to SAMPLE`
 - ❖ In the DB2 Command Window, all commands to DB2 must begin with `db2`.
6. Execute the DDL to create the tables by issuing the following command:
`db2 -t -f META-INF\Table.ddl`

Note that this method of finding and extracting the table creation DDL will work for any database you choose in the AAT's code generation dialog. Thus if you are using Oracle or Sybase to execute these examples, you can pull the proper creation DDL from this file in the same way.

Step 7: Deploy the .ear file into WebSphere AEs

Now that you have generated all the appropriate client code and created the necessary database tables you are ready to deploy the new EJBs. As was discussed earlier, in all previous exercises you have deployed the EJB *.jar* files to the WebSphere application server, and generated the *.ear* file only to create deployment code inside it that would be used by application clients. In this case, though, to allow the EJBs to reference each other when the module visibility level is set to "Application," you will have to deploy the *.ear* file, rather than each individual *.jar* file.

In fact, it's slightly more problematic than that – there is a minor bug in the current version of WebSphere that will require you to install the "deployed" version of the *.ear* file. For some reason the paths necessary to compile the generated code are not correctly resolved by the Administrative Console's deployment tool, although they are by the deployment tool that is part of the AAT.

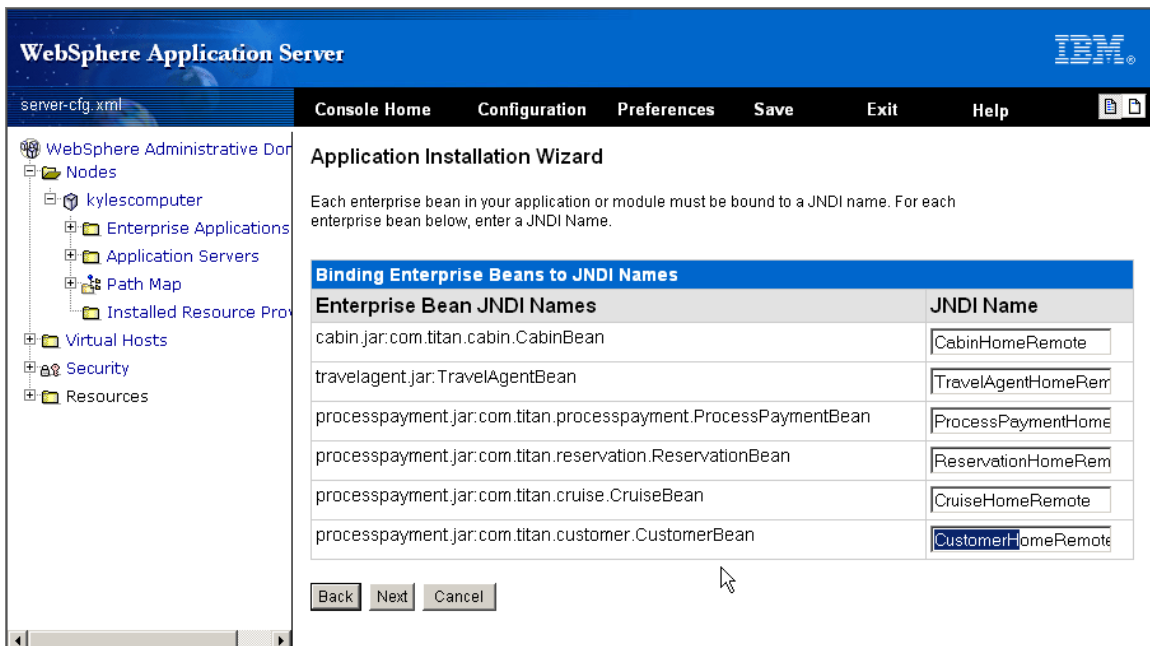
- ❖ This bug will be fixed in a later update of WebSphere. The best practice would be to deploy the "undeployed" version of the *.ear* if it were possible.

Begin this step as you have in all of the other exercises. Start the application server and ensure that it has completed its startup routine. Open an Administrative Console and under **Nodes** navigate down to the **Enterprise Applications** folder. Select that folder to start the Application Installation wizard. This time, however, instead of typing the path to the EJB *.jar* file into the Path field, or using the Browse button to select the EJB *.jar* file, select the *Chapter12/Exercise12-1/dev/Deployed_processpayment.ear* file you generated in Step 5.

From that point on you can proceed to the rest of the wizard pages as you would normally. You will need to be aware of two pages that are new or different, however.

When you arrive at the wizard page that allows you to set the JNDI names of the EJBs, you will see that it contains entries for all of the EJBs in the `.ear` file, rather than the single EJB you have deployed in all other exercises:

Figure 81: Deploying multiple EJBs from an `.ear` file



Set the JNDI names for the EJBs exactly as in the following table:

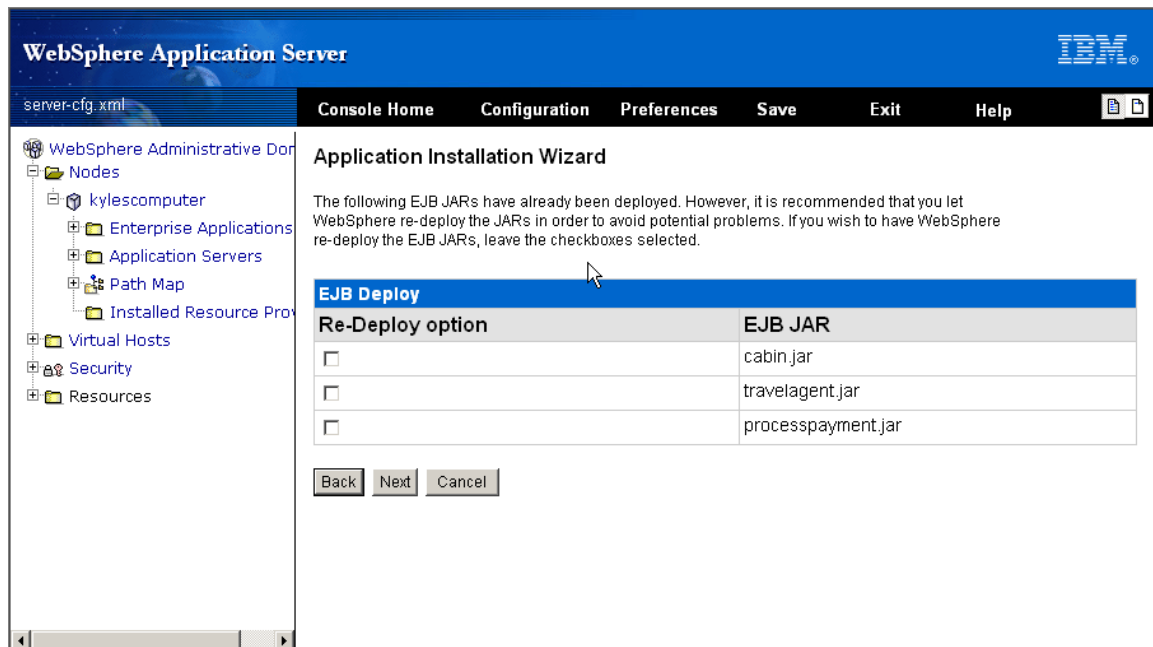
<i>Class</i>	<i>JNDI Name</i>
Cabin.jar: com.titan.cabin.CabinBean	CabinHomeRemote
TravelAgent.jar: TravelAgentBean	TravelAgentHomeRemote
Processpayment.jar: com.titan.processpayment.ProcessPaymentBean	ProcessPaymentHomeRemote
Processpayment.jar: com.titan.reservation.ReservationBean	ReservationHomeRemote
Processpayment.jar: com.titan.cruise.CruiseBean	CruiseHomeRemote
Processpayment.jar: com.titan.customer.CustomerBean	CustomerHomeRemote

Later you will see a similar page that allows you to set the binding of each of the CMP EJBs to data sources. Just set all of the EJB `.jar` default **Data Source** reference fields to be **jdbc/SampleDB**. Likewise, you will see a wizard page that allows you to set the resource reference to `jdbc/titandb`. Again, define this reference as **jdbc/SampleDB**.

You will also see a page that asks you to define the reference to the Cabin EJB in the `TravelAgent.jar` file (`ejb/CabinHomeRemote`). Set the value of that reference to be **CabinHomeRemote**.

Finally, just before the end of the deployment process you will see the following page:

Figure 82: Setting the redeployment option



It is very important that you “uncheck” all the **Re-deploy option** checkboxes before pressing **Next** to move to the final page of the wizard. If these boxes remained checked, the Application Installation Wizard would attempt to regenerate all of the deployment code in each `.ear` file – and would fail in the case of `processpayment.jar`, because of the bug in the deployment wizard we described earlier.

Step 8: Test the new EJBs with the test client

Launch the test client with

```
launchClient Deployed_processpayment.ear
```

This client doesn't do much to verify that the ProcessPayment EJB is working. You just want to verify that the client runs without displaying any error messages. If you really want to verify that the application works correctly, after you run it open a DB2 CLP, connect to the [SAMPLE](#) database, and issue the command to query the [PAYMENT](#) table:

```
select * from payment
```

If you see any rows displayed, then the example client worked correctly.

Exercise 12.2: A Stateful Session Bean

In this exercise you will deploy a new version of the TravelAgent EJB. This particular exercise is interesting because you will learn about a feature of WebSphere AEs that you haven't explored yet – the command-line deployment tool SEAppInstall, which can be used as an alternative to the Administrative Console for deploying and undeploying applications.

Step 1: Compile the Source Files

To compile the class files for this exercise, open a command prompt, change to the directory *Chapter12/Exercise12-2/dev*, and execute *TravelAgentBeanCompile.bat*. After the batch file executes, verify that there are *.class* files for all of the *.java* files in the directory structure rooted at */dev/com/titan*.

Step 2: Assemble the EJB .jar file

To build the *travelagent.jar* EJB *.jar* file, execute the *BuildTravelAgentJarFile.bat* batch file in the *Chapter12/Exercise12-1/dev* directory. This batch file will assemble the compiled classes and the *ejb-jar.xml* file appropriately.

Step 3: Create an application client .jar file

Just as in the previous exercise, you will have to build an example program to demonstrate how the TravelAgent EJB functions, because the EJB book does not provide one. You will notice, however, that most of the code in this example is taken from the bits and pieces of the applet example in Chapter 12 of the EJB book.

The following program, *Chapter12/Exercise12-2/dev/com/titan/ReservationApplication.java*, will test some of the features of the new version of TravelAgent EJB and will help you determine whether it has been correctly deployed to WebSphere AEs:

```
package com.titan.travelagentclient;

import com.titan.travelagent.*;
import com.titan.customer.*;
import com.titan.processpayment.*;
import com.titan.reservation.*;
import com.titan.cruise.*;
import javax.naming.*;
import java.util.*;
import java.text.*;
import javax.rmi.PortableRemoteObject;
```

```
public class ReservationApplication {

    public static void main(String args[]) {
        try {
            Context jndiContext = getInitialContext();

            /* Create a new Customer. The current code doesn't
             * delete this customer at the end, so you must remove
             * it from the DB manually after running the program.
             */
            Object ref = jndiContext.lookup("CustomerHomeRemote");
            CustomerHomeRemote customerHome = (CustomerHomeRemote)
                PortableRemoteObject.narrow(
                    ref, CustomerHomeRemote.class);
            String ln = "Public";
            String fn = "John";
            String mn = "Q";
            int nextID = 2;
            CustomerRemote customer =
                customerHome.create(nextID, ln, fn, mn);

            /* Create a new Travel Agent servicing this customer.
             */
            Object tref =
                jndiContext.lookup("TravelAgentHomeRemote");
            TravelAgentHomeRemote home =
                (TravelAgentHomeRemote)
                    PortableRemoteObject.narrow(
                        tref, TravelAgentHomeRemote.class);
            TravelAgentRemote agent = home.create(customer);

            /* Create a new Cruise and connect the customer
             * to the Cruise through this travel agent.
             */
            Integer cruise_id = new Integer(1);
            Object cref = jndiContext.lookup("CruiseHomeRemote");
            CruiseHomeRemote chome =
                (CruiseHomeRemote)
                    PortableRemoteObject.narrow(
                        cref, CruiseHomeRemote.class);
            chome.create(1, "Bahamas paradise", 1);
            agent.setCruiseID(cruise_id);

            /* Set the Cabin ID to a Cabin created in Ex. 4.1
             */
        }
    }
}
```

```
        Integer cabin_id = new Integer(1);
        agent.setCabinID(cabin_id);

        /* Create a CreditCardDO with all the fields.
        */
        String cardNumber = "12345678901234";
        DateFormat dateFormatter =
            DateFormat.getDateInstance(DateFormat.SHORT);
        Date date = dateFormatter.parse("1/1/2002");
        String cardBrand = "Amex";
        CreditCardDO card =
            new CreditCardDO(cardNumber, date, cardBrand);
        double price = 1000.0;

        /* Book passage for this customer and print the ticket.
        */
        TicketDO ticket = agent.bookPassage(card, price);
        System.out.println(ticket);

        /* This is a good place to clean things out. You need to
        * delete the Customer, Cruise Payment, and Reservation
        * if you want to run the example again.
        */

    } catch (Exception e) {
        System.out.println("Exception caught " + e);
        e.printStackTrace();
    }
}

public static Context getInitialContext()
    throws javax.naming.NamingException {

    java.util.Properties properties =
        new java.util.Properties();
    properties.put(
        javax.naming.Context.PROVIDER_URL, "iiop:///");
    properties.put(
        javax.naming.Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
    InitialContext initialContext =
        new InitialContext(properties);
    return initialContext;
}
}
```

Use the AAT wizards to build the application client *.jar* files for this example program. Select the **Create Application Client** wizard to start the process. Enter the following information to create your application client:

- ◆ **File Name:** **ReservationApplication.jar**
- ◆ **Display Name:** **ReservationApplication**
- ◆ **Files to import:** **com.titan.travelagentclient.ReservationApplication.class** (select *Chapter12/Exercise12-2/dev*)
- ◆ **Main Class:** **com.titan.travelagentclient.ReservationApplication.class**
- ◆ No icons, EJB references, resource references, or environment entries

When you have finished, save the application client as *dev/ReservationApplication.jar*.

Step 4: Create the .ear files

Use the AAT wizards to assemble the application client *.jar* file and the EJB *.jar* file you just created, the *cabin.jar* file created in Exercise 4.1, and the *processpayment.jar* file created in the previous exercise into a new *.ear* file, *dev/travelagent.ear*.

Then select the AAT's **Generate Deployed Code** menu option to begin generating a deployed version of the *.ear*.

Step 5: Use SEAppInstall to deploy the new TravelAgent .ear file

Now you are ready to begin learning how to use SEAppInstall to deploy applications into WebSphere. The first step you have to take will show how different using the command-line tools is from using the Web-based administration tools. Begin by going to the First Steps application and shutting down the application server.

Now, why do you want to do that? Up to this point you have always needed to make sure the application server *was* running before you could use the Administrative Console, because the Administrative Console servlets and JSPs run within a web application contained in the application server. By contrast, the SEAppInstall program does not run within the application server, and does not require it to be running. In fact, it's best to shut down the application server before using SEAppInstall, although it is not required.

Open a command prompt and `cd` to the *Chapter12/Exercise12-2/dev* directory. The first feature of SEAppInstall that you want to investigate is the list capability. You can use this feature to obtain a list of installed applications. Type the following at the command prompt:

```
SEAppInstall -list apps
```

You should see output like the following:

```
IBM WebSphere Application Server Release 4, AEs
J2EE Application Installation Tool, Version 1.0
Copyright IBM Corp., 1997-2001
```

```
The -configFile option was not specified. Using
C:\WebSphere\AppServer\config\server-cfg.xml
Loading Server Configuration from
C:\WebSphere\AppServer\config\server-cfg.xml
Server Configuration Loaded Successfully
```

Installed Applications

- 1) sampleApp
- 2) Server Administration Application
- 3) WebSphere Application Server Samples
- 4) petstore
- 5) Ship.ear
- 6) BMPShip
- 7) ProcessPayment

What you now want to do is to remove the *processpayment* application and replace it with the new *travelagent* application you created in the preceding step. You will remember that this *.ear* file contains, not only all of the same EJB code that was in the *processpayment* application, but also the newly extended TravelAgent EJB.

To uninstall the *processpayment* application, type the following at the command prompt:

```
SEAppInstall -uninstall ProcessPayment -delete true
```

You should see something like the following output:

```
IBM WebSphere Application Server Release 4, AEs
J2EE Application Installation Tool, Version 1.0
Copyright IBM Corp., 1997-2001
The -configFile option was not specified. Using
C:\WebSphere\AppServer\config\server-cfg.xml
Loading Server Configuration from
C:\WebSphere\AppServer\config\server-cfg.xml
Server Configuration Loaded Successfully
Removed Application From Server: ProcessPayment
Deleted All Files for Application ProcessPayment from Server
Backing up Server Configuration to
C:\WebSphere\AppServer\config\server-cfg.xml
Saving Server Configuration to
C:\WebSphere\AppServer\config\servercfg.xml
Save Server Config Successful
```

Now you are ready to install the new application. This is slightly more complicated than the `list` or the `uninstall` because you must provide the binding information that you previously have typed into the web administration console. Here you will use `SEAppInstall` in what is called “interactive” mode, typing information into the prompts that `SEAppInstall` provides.

Begin by typing the following at the command line:

```
SEAppInstall -install travelagent.ear
```

You will see the following output, ending in a prompt:

```
IBM WebSphere Application Server Release 4, AEs
J2EE Application Installation Tool, Version 1.0
Copyright IBM Corp., 1997-2001
The -configFile option was not specified. Using
C:\WebSphere\AppServer\config\server-cfg.xml
You have chosen to install this application
in interactive mode.
When prompted for information, pressing ENTER without entering
any information will cause the default value (shown in [])
to be used for that property.
When prompted for information, entering ! as the value will cause
the current value to be erased. This can be used to unset current
information.
Loading Server Configuration from
C:\WebSphere\AppServer\config\server-cfg.xml
Server Configuration Loaded Successfully
Loading H:\EJBWorkbook\Source\Chapter12\Exercise12-
2\dev\travelagent.ear
Getting Expansion Directory for EAR File
Expanding EAR File to
C:\WebSphere\AppServer\installedApps\travelagent.ear
Do you wish to deploy all of the EJBs in this application
([Y]es/[n]o)?
```

Here you have the choice either to deploy or not to deploy your EJBs. If you were using `SEAppInstall` to install an `.ear` file for which you had previously generated deployment code inside AAT, you could choose **no**. In this case, however, you want to see how the command line-tool deploys EJBs, so instead type **y** and press **Enter**. You'll see the following text, followed by a prompt:

```
The following Database Types are supported for EJB Deployment
-----
0) Not Applicable
1) IBM DB/2 Universal Database, Version 7.1 FP3, or higher
2) IBM DB/2 Universal Database for OS/390, Version 6
3) IBM DB/2 Universal Database for OS/400, Version 4 Release 5
4) Informix Database, Version 9.2, or higher
```



```
5) Microsoft SQL Server, Version 7, or higher
6) Oracle 8i
7) SQL 92 Compliant Database Server
8) SQL 99 Compliant Database Server
9) Sybase Database, Version 11.92, or higher
10) MySQL Database, Version 3.2.3
```

```
Which type of database are you using (optional. specify the number)?
```

In this case, choose the database type you are using. If you are using DB2, type **1** and press **Enter** to see the next prompt:

```
What DB Schema name do you want to use for this application
(optional)?
```

Here simply press **Enter**, because you do not want to provide a schema name. (You have not used a schema name for any of the other examples). SEAppInstall will begin generating the EJB deployment code. Just as with the AAT or the web console, this could take a few minutes. You will see a **lot** of output (in fact, it will probably scroll past the end of your command prompt buffer), so we won't include it all here. Just be certain to keep an eye out for the following message, which will indicate the end of processing for each *.jar* file:

```
[EJBDeploy] EJBDeploy complete.
[EJBDeploy] 0 Errors, 0 Warnings, 0 Informational Messages
```

You should see that message repeated three times, once for each EJB *.jar* file. If you see instead any number of errors other than zero, you might need to go back and double-check that you have followed the previous steps correctly.

- If you do see any errors, it is OK to press Control-C to cancel the SEAppInstall batch file and then come back and repeat the step later. SEAppInstall will clean up any temporary directories it needs before using them.

You will know that deployment code generation is complete when you see a display containing the following information and prompt (you will see differences in the structures of temporary directories, of course, but you can safely ignore them):

```
[EJBDeploy] 0 Errors, 0 Warnings, 0 Informational Messages
[EJBDeploy] Delete undeployed module file
C:\DOCUME~1\kbrown\LOCALS~1\Temp\travelagent_ear\
processpayment_source.jar
[EJBDeploy] Save the deployed modules as EAR file
C:\DOCUME~1\kbrown\LOCALS~1\Temp\deployed_travelagent.ear
EJB Deployment Completed Successfully. Reloading deployed EAR file
[C:\DOCUME~1\kbrown\LOCALS~1\Temp\deployed_travelagent.ear].
Loading C:\DOCUME~1\kbrown\LOCALS~1\Temp\deployed_travelagent.ear

Please provide the following EJB Jar Binding Information
-----
```

```
EJB Jar: travelagent.jar
-----
```

```
Default Datasource JNDI Name (optional) []:
```

You are now ready to start entering binding information for the *.ear* file. The first prompt asks you for the default data source JNDI name. Type **jdbc/SampleDB** and press **Enter**. Then press **Enter** twice more, once for the *UserId* and once for the *Password*.

Next SEAppInstall will ask you for information about the first EJB in the *.jar* file:

```
Enterprise Bean: com.titan.travelagent.TravelAgentBean
-----
```

```
JNDI Name []:
```

Type **TravelAgentHomeRemote** and press **Enter**.

Repeat the same process at the next prompt, which asks for the information on *Cabin.jar*. Type in **jdbc/SampleDB** as the default data source for the *.jar* file, and press **Enter** for the *UserId* and *Password*. Likewise type **CabinHomeRemote** as the JNDI name of the enterprise bean, and press **Enter** when prompted for a default data source name for the Cabin EJB.

Next you will be asked to provide information for the *processpayment.jar* file. Type in **jdbc/SampleDB** as the default data source for the *.jar* file, then type in the EJB names as you see them here (at each colon type what is shown in bold text and press **Enter**; if nothing follows the colon, just press **Enter**):

```
Enterprise Bean: com.titan.processpayment.ProcessPaymentBean
-----
```

```
JNDI Name []: ProcessPaymentHomeRemote
```

```
Enterprise Bean: com.titan.reservation.ReservationBean
-----
```

```
JNDI Name []: ReservationHomeRemote
```

```
Datasource JNDI Name []:
```

```
Enterprise Bean: com.titan.cruise.CruiseBean
-----
```

```
JNDI Name []: CruiseHomeRemote
```

```
Datasource JNDI Name []:
```

```
Enterprise Bean: com.titan.customer.CustomerBean
-----
```

```
JNDI Name []: CustomerHomeRemote
```

```
Datasource JNDI Name []:
```

After you finish setting the default data sources and JNDI bindings for the EJBs, SEAppInstall will ask you to provide information for the resource references in the *.ear* file. You will see the following output, followed by a prompt:

```
The following Resource Factories are available on this node.
```

- ```

1) jdbc/Session: Data source for session persistence
2) jdbc/SampleDB: New DB2 Data source
3) jdbc/SampleDataSource: Increment Bean Datasource
4) jdbc/sample: Samples Gallery IDB Datasource
5) ps/PetStoreDatasource: Petstore IDB Datasource
```

```
For each of the following Resource Ref, please specify a
JNDI Name and a default User ID and Password
```

- ```
-----  
1) jdbc/titanDB  
(travelagent.jar:com.titan.travelagent.TravelAgentBean)  
JNDI Name []:
```

Just as in previous exercises, you want to map *jdbc/titanDB* to the SampleDB data source, so type **jdbc/SampleDB** at this prompt and press **Enter**. Because that resource reference is defined in two different *jar* files in this *.ear* file, immediately after your response to this prompt the program will ask for this mapping information a second time. Again, type **jdbc/SampleDB** and press **Enter**.

After completing the resource reference bindings, SEAppInstall will move on to the EJB reference bindings. You will see the following output, followed by a prompt:

```
The following EJBs are available on this node.  
Format: n) <Bean Name> [<JNDI Name>]  
-----  
1) com.titan.travelagent.TravelAgentBean [TravelAgentHomeRemote]  
2) com.titan.cabin.CabinBean [CabinHomeRemote]  
3) com.titan.processpayment.ProcessPaymentBean  
[ProcessPaymentHomeRemote]  
4) com.titan.reservation.ReservationBean [ReservationHomeRemote]  
5) com.titan.cruise.CruiseBean [CruiseHomeRemote]  
6) com.titan.customer.CustomerBean [CustomerHomeRemote]
```

```
Please Specify a JNDI Name for the following EJB References  
-----
```

- ```
1) ejb/ProcessPaymentHomeRemote
(travelagent.jar:com.titan.travelagent.TravelAgentBean) []:
```

At this prompt, type **ProcessPaymentHomeRemote** and press **Enter**. Likewise, you will have to enter the following mappings (again, type the names you see in bold text and press **Enter** after each):

- ```
2) ejb/CabinHomeRemote  
(travelagent.jar:com.titan.travelagent.TravelAgentBean) []:  
CabinHomeRemote
```

```
3) ejb/CruiseHomeRemote
(travelagent.jar:com.titan.travelagent.TravelAgentBean) []:
CruiseHomeRemote
4) ejb/CustomerHomeRemote
(travelagent.jar:com.titan.travelagent.TravelAgentBean) []:
CustomerHomeRemote
5) ejb/ReservationHomeRemote
(travelagent.jar:com.titan.travelagent.TravelAgentBean) []:
ReservationHomeRemote
```

SEAppInstall will then need to set up the security bindings for your application. You will see the following output:

```
Please Specify a Subject Name for the following Security Roles
Each Role takes a comma separated list of Users/Groups/Specials
A user is specified as U:<user name>
A group is specified as G:<group name>
A special subject is specified as S:EVERYONE or S:ALLAUTHUSERS or
S:AllAuthenticatedUsers
Example: U:bob,G:Power Users
-----
1) everyone []:
```

Type **S:EVERYONE** at the prompt and press **Enter**. You will then see the following question:

```
No "Run As Roles" Mappings defined
travelagent.jar contains method permissions. Do you wish to deny
access to all unprotected methods ([Y]es/[n]o)?
```

Type **n** and press **Enter**. At this point you should see the following output, indicating that you have successfully finished installing the *.ear* file.

```
All unprotected methods will remain unprotected.
All unprotected methods will remain unprotected.
All unprotected methods will remain unprotected.
Installed EAR On Server
Validating Application Bindings...
CHKW4518W: No datasource has been specified for the container
managed entity bean ?. The default datasource specified for the EJB
jar will be used.
CHKW4518W: No datasource has been specified for the container
managed entity bean ?. The default datasource specified for the EJB
jar will be used.
CHKW4518W: No datasource has been specified for the container
managed entity bean ?. The default datasource specified for the EJB
jar will be used.
```

```
CHKW4518W: No datasource has been specified for the container
managed entity bean ?. The default datasource specified for the EJB
jar will be used.
Finished validating Application Bindings.
Saving EAR File to directory
Saved EAR File to directory Successfully
Backing up Server Configuration to
C:\WebSphere\AppServer\config\server-cfg.xml~

Saving Server Configuration to C:\WebSphere\AppServer\config\server-
cfg.xml
Save Server Config Successful
JSP Pre-compile Skipped.....
Installation Completed Successfully
```

If you do not see “Installation Completed Successfully” at the end, you will want to use the `-uninstall` option to remove the application, go back over the previous steps, and try again.

If you are interested in understanding more about the other options and capabilities of SEAppInstall, refer to the InfoCenter. As to which tool you should use to administer WebSphere, the web-based console or SEAppInstall; that is largely a matter of personal preference. It is probably faster to use SEAppInstall for removing applications and finding out what applications are installed, but when you want to install new applications there is little difference between using SEAppInstall in interactive mode and using the Administrative Console.

Step 6: Launch the application client

At this point you can test your new code with the following command:

```
launchClient Deployed_travelagent.ear
```

Finally, verify that your output matches the following:

```
IBM WebSphere Application Server, Release 4.0
J2EE Application Client Tool, Version 1.0
Copyright IBM Corp., 1997-2001

WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment
has completed.
WSCL0014I: Invoking the Application Client class
com.titan.travelagentclient.ReservationApplication
John Public has been booked for the bahamas cruise on ship 1.
Your accommodations include Master Suite a 3 bed cabin on deck level
1.
Total charge = 1000.0
```

Congratulations! You've now finished the WebSphere AEs EJB workbook. You can deepen your understanding and sharpen your skills by going back and adding some of the improvements to the client programs that we suggested along the way, and trying out your own variations. We hope you've enjoyed using WebSphere AEs and are ready to tackle building your own EJB programs with the platform.

About the Author

Kyle Brown is an Executive Java Consultant with IBM WebSphere Services. He has over 11 years of experience in designing and architecting large-scale OO systems. He specializes in developing and promoting “best practices” approaches to designing large-scale systems using Java 2 Enterprise Edition (J2EE) and IBM’s WebSphere product suite. He is the author of *Enterprise Java Programming with IBM WebSphere* (Addison-Wesley, 2001), a well-known conference speaker, and an internationally recognized expert in J2EE and WebSphere.

About the Series

Each of the books in this series is a server-specific companion to the third edition of Richard Monson-Haefel’s best-selling and award-winning *Enterprise JavaBeans* (O’Reilly 2001), available at <http://www.titan-books.com/> and at all major retail outlets. It guides the reader step by step through the exercises called out in that work, explains how to build and deploy working solutions in a particular application server, and provides useful hints, tips, and warnings.

These workbooks are published by Titan Books in the context of a friendly agreement with O’Reilly and Associates, the publishers of *Enterprise JavaBeans*, to provide serious developers with the best possible foundation for success in EJB development on their chosen platforms.

Colophon

This book is set in the legible and attractive Georgia font. Manuscripts were composed and edited in Microsoft Word, and converted to PDF format for download and printing with Adobe Acrobat.