



# Auction Scenario: B2C Online Auction Site Using EJB Tools

**Note!**

Before using this information and the product it supports, be sure to read the general information under **"Notices"** on page 55.

**Seventh Edition (December 2003)**

**© Copyright International Business Machines Corporation 2002 . All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

## Chapter 1. Overview of the Online

### Auction application . . . . . 1

J2EE applications . . . . . 1

The Online Auction sample application . . . . . 3

    Application design . . . . . 3

## Chapter 2. Preparing for the Online

### Auction application . . . . . 5

System prerequisites . . . . . 5

Setting up the development and run-time  
environments . . . . . 5

    Install and configure the CVS server (optional) . . 6

    Install WebSphere Studio . . . . . 6

    Enable the JDBC 2.0 driver and create the  
    database . . . . . 6

    Create the database tables . . . . . 8

## Chapter 3. Working with the Online

### Auction application . . . . . 11

Running the ready-made sample application . . . 12

Part 1 — Creating projects, CMP beans, and EJB  
queries . . . . . 12

    Create the required projects . . . . . 12

    Add CMP beans to the EJB project . . . . . 13

    Adding EJB queries. . . . . 19

    Generating deployment code . . . . . 22

Part 2 — Creating session beans and  
message-driven beans . . . . . 23

    Creating session beans. . . . . 24

    Add the business logic to the session bean . . . 26

    Creating message-driven beans . . . . . 29

Part 3 — Testing enterprise beans on a server,  
creating a Web service and running the application . 31

    Create a server and add a data source . . . . . 32

    Test the business methods . . . . . 35

    Create the Web Service . . . . . 39

    Set up the team environment . . . . . 42

    Version the application . . . . . 44

    Add shared projects to another workspace . . . 44

    Add a professional Web front end . . . . . 49

    Debug the application . . . . . 51

## Notices . . . . . 55

Programming interface information . . . . . 57

Trademarks and service marks . . . . . 57



---

## Chapter 1. Overview of the Online Auction application

---

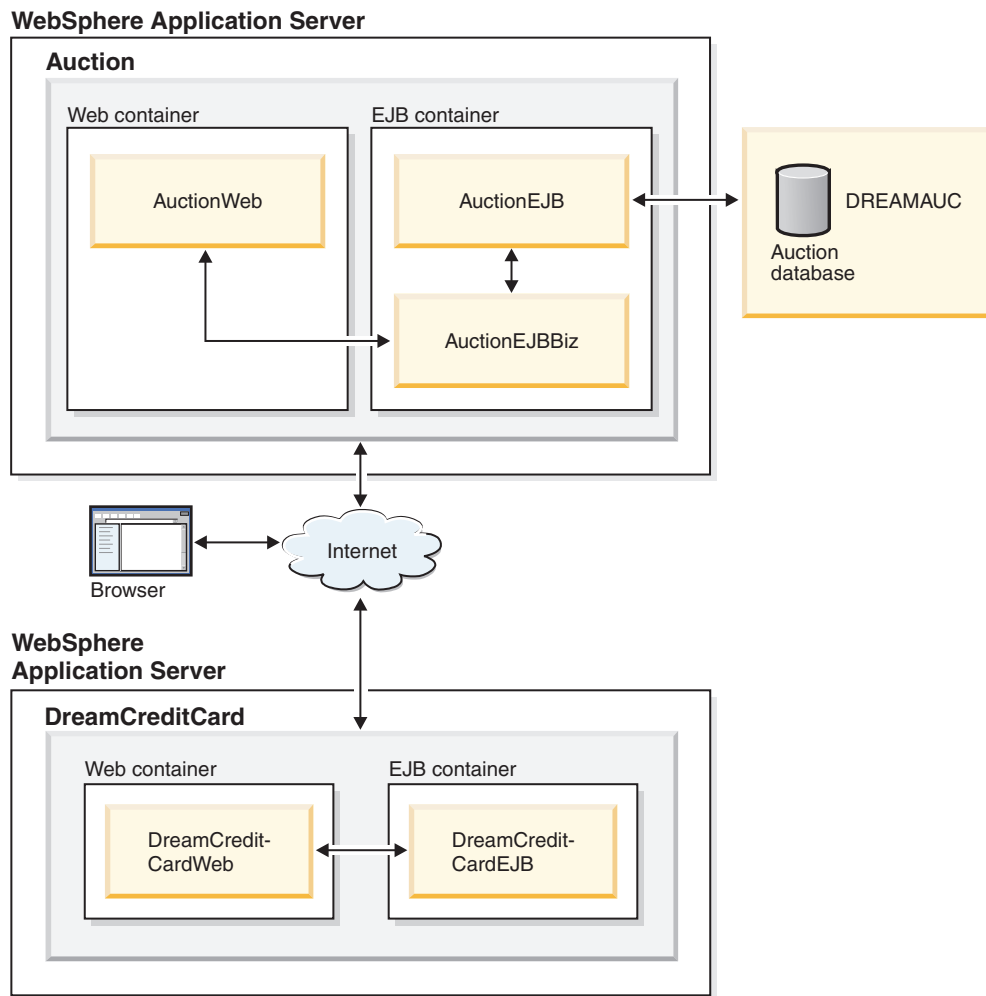
### J2EE applications

Although many organizations continue to use Web-centric e-business applications, these applications are limited and must often be re-engineered if they are called upon to perform more complex tasks. For this reason, many organizations have elected to build multi-tiered Java™ 2 Platform, Enterprise Edition (J2EE) applications, which are scalable, transactional, extensible, and secure. J2EE application components are separated into presentation, logic, and data layers. This enables organizations to reuse more code, which saves them time and money. J2EE applications can also help organizations provide customers with consistent, timely and convenient access to Web services. And organizations can use J2EE applications to leverage new Web and wireless technologies, which opens up new market opportunities.

EJB technology is the framework on which the J2EE specification is based. It is used to develop and deploy enterprise beans, which are server-side software components that contain an application's business logic. Applications developed using EJB technology generally consist of multiple tiers and include the following application elements:

- A Web browser
- A Web container, which contains the application's Web components, such as servlets and JavaServer Pages (JSP) files. These Web components handle the application's presentation logic. The JSP files, for example, provide the client with dynamic Web content. The complexities of security, transaction management, resource pooling, and lifecycle management are handled automatically by containers.
- An EJB container, which contains enterprise beans and supporting components that access Enterprise Information System (EIS) resources in response to requests from the Web container.
- An EIS, such as a relational database.

The following figure shows a typical multi-tiered topology for an EJB application:



J2EE applications provide many advantages in addition to their scalability. For example, they enable you to do the following:

- Rapidly develop simple thin application clients and integrate them with existing applications and databases, which helps protect your investment in legacy systems.
- Clearly separate the front-end user experience from back-end data access, which allows development roles to be more clearly defined in building EJB applications.
- Employ advanced EJB technology to manage system-level services for your application, such as transactions, security, client connectivity, and database access. This allows you to focus on the business logic for your applications, which reduces the cost of application development and maintenance.

Even small businesses can benefit from the many advantages of using EJB technology in their e-business applications. And if their business grows, their EJB applications can easily accommodate the growth where other types of applications might eventually prove inadequate.

---

## The Online Auction sample application

By following the instructions in this document, you learn how to use WebSphere® Studio by building an Online Auction sample application and running it in a J2EE-compliant server environment. As the name implies, the sample application features an auction Web site, where you can browse auction items, submit items for sale, and bid on items. To save you time, a number of application components are already provided for you, such as images and code fragments that you can copy and paste into existing source code. This enables you to focus on learning this product rather than spending your time performing the more tedious tasks associated with developing an application.

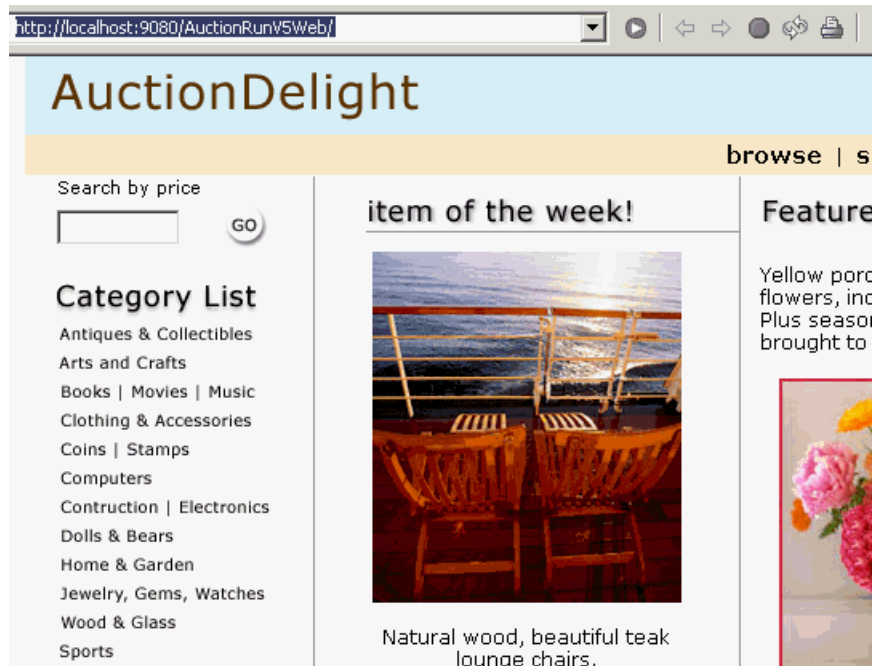
Since the online auction sample application employs J2EE technology, it features a wide variety of application components, such as Java beans, JDBC, HTML, JavaScript™, servlets, JSP pages, GIF files, and enterprise beans. You can develop or enhance all of these component types using WebSphere Studio, which provides multiple application development tools, such as local and remote environments that are based on WebSphere Application Server. The Online Auction sample application complies to the J2EE 1.3 specification, using EJB 2.0 beans. To develop this application, you will use the EJB tools, the Server tools, the Web services tools, and the CVS team environment. You can also use the UML class diagram editor for EJB and Java elements to develop parts of the application. The application consists of business logic contained in enterprise beans that access a database and presentation logic provided by various Web resources.

### Application design

The design for the Auction application uses a pattern called Session Facade. This practice of wrapping an application's entity bean layer in a layer of session beans is an EJB design best practice. This design hides the complexity of the entity beans, and reduces the number of remote calls from the client by using local interfaces to access the entity beans. This loose coupling between the client and the entity beans also increases reuse and maintainability. There is a clean separation of business logic from the presentation layer. EJB clients should never access entity beans directly but go through session beans instead. In this application, a session bean called ItemHelper allows clients to update the database. The updates will be performed using six entity beans: AccountsPayable, Address, Bid, Onlineitem, Registration, and Sale. These beans map to the database tables ACCOUNTSPAYABLE, ADDRESS, BID, ONLINEITEM, REGISTRATION, and SALE, respectively. They are located in the AuctionEJB project.

The AuctionEJBBiz project contains the business logic of the application. It consists of four session beans and one message-driven bean:

- The ItemHelper session bean creates new items (Onlineitem and Sale entity beans), finds existing items (Onlineitem entity bean), and processes expired items (Sales, Onlineitem, Bid, and Registration entity beans).
- The BidHelper session bean places bids (Bid and Onlineitem entity beans) and sends messages to the BidProxy message-driven bean when a user bids on an item.
- The DBConfig session bean reserves the keys for the database tables and resets the database using a JDBC connection.
- The RegistrationHelper session bean retrieves a list of all registered users of the auction site (Registration entity bean).
- The BidProxy message-driven bean receives bids and bids on the user's behalf.



The user opens the Auction Web site (index.jsp) in a browser. The user selects a category of auction items, which sends a request to the Web server and returns a JSP page (FloralResults.jsp). To sell an item, the user selects "sell" from the navigation bar. This opens the sale form (Sell.jsp). The user then enters the values for the item for sale, such as value, description, and length of auction. The user submits the form and the data is transmitted to the database via the create method of the session bean (ItemHelper). The Floral page reopens displaying the name of the item the user registered in the auction. To search the site for items by price, the user enters a price in the search field. The findByValue method of the session bean searches the database for items in that price range and updates FloralResults.jsp with those items. To bid on an item, the user selects the desired item and opens a bid form (FloralDetails.jsp). The user can either bid by the next increment amount, or enter a maximum bid, allowing the auction site to bid on the user's behalf. The user clicks Submit and registers the bid in the auction. The Web server refreshes the page and displays the new bid (FloralResults.jsp). When the items have expired, a Web service processes the winning bidder's credit card information.

Although you can walk through the Online Auction sample application without any prior experience with WebSphere Studio, you will find it helpful to browse the *Getting Started* book before working with the sample application. You can find a link to *Getting Started* in the online help for the WebSphere Studio development environment that you have installed. This will give you a basic familiarity with WebSphere Studio. If you need detailed information about any of the WebSphere Studio tools, you can also consult the WebSphere Studio online help.

To work with the Online Auction sample application, you complete one or both of the following activities:

- Run the ready-made Online Auction sample application
- Build the Online Auction sample application for yourself

Before you can perform either of these activities, you must complete the tasks in the topic "Preparing for the Online Auction sample application".



---

## Chapter 2. Preparing for the Online Auction application

To build the Online Auction application, you need to ensure that all system prerequisites are met and that the development and run-time environments are correctly installed and configured.

---

### System prerequisites

The hardware and operating system prerequisites for building and running the Online Auction application are the same as those that are specified for the WebSphere Studio product. This scenario was originally developed for WebSphere Studio Application Developer. The instructions provided here should work for other WebSphere Studio product configurations that include the EJB tools.

This application requires that the WebSphere Studio v5.1 test environment is installed. If you did not select this option when you installed this product, you can install this feature with the WebSphere Studio installer.

In addition to WebSphere Studio and its prerequisites, you need a back-end database installed in order to build and run the Online Auction application. You can use one of the following databases:

- DB2 Universal Database™ (UDB) for Windows® Version 7.2 FP7 (or later) or DB2 Universal Database (UDB) for Linux Version 7.2 FP7 (or later).
- DB2 Universal Database (UDB) for Windows Version 8.1 FP3 (or later) or DB2 Universal Database (UDB) for Linux Version 8.1 FP3 (or later). DB2 is included with most configurations of the WebSphere Studio product family. If DB2® was not included in your WebSphere Studio package, you can download it from [www.ibm.com/software/data/db2](http://www.ibm.com/software/data/db2).
- Cloudscape™, V5.1. Cloudscape is an embedded Java database that is included with this product. No installation is required.
- DB2 Universal Database for iSeries™ V5R1.

**Note:** DB2 UDB 7.2 and 8.1 should be installed on the same machine as WebSphere Studio.

The optional software is as follows:

- The latest version of Concurrent Versions System (CVS), available from <http://www.cvshome.org/>.

---

### Setting up the development and run-time environments

In preparation for building and running the Online Auction application, you need to perform the following activities, which are described in the sections that follow.

- Install and configure the CVS server (optional).
- Install WebSphere Studio as your development environment (required).
- Install your database and run the database script to create the database tables (required).

## Install and configure the CVS server (optional)

In this section, you get some tips on how to install the CVS server and set up the repository for team development. The installation of CVS is optional for the application, but there are sections in the instructions that explain how to use it with WebSphere Studio for team programming. You can download a free copy of CVS from [www.cvshome.org](http://www.cvshome.org).

To install and configure the CVS server:

1. Install the CVS product using the product ZIP file. If you are planning to use it for production purposes, it is recommended that you install the product on a Linux server.
2. Add the repository (or repositories) by creating a directory and running the **cvs init** command.
3. Authorize the users. It is recommended that you use an SSH connection method and create a user ID on the server for each developer.
4. Configure the server to run the SSH daemon by default.

More detailed information about installing, configuring, and using the CVS server can be found in the CVS manual and in articles located on the WebSphere Developer Domain.

## Install WebSphere Studio

Install WebSphere Studio using the installation instructions that accompany the product.

## Enable the JDBC 2.0 driver and create the database

**Note:** Skip this section if you are using Cloudscape or iSeries. The following instructions apply to DB2 only.

In this section, you perform the following activities:

- Change your DB2 installation to use the JDBC 2.0 driver
- Create the sample database

To enable the JDBC 2.0 driver and create the database:

### Windows

1. If your database is not already installed, install it by running the installation program.
2. This step applies to DB2 7.2 only. DB2 8.1 does not require this step. Change your DB2 installation to use JDBC 2.0 by completing the following steps:
  - a. Open a DB2 command window (**Start > Programs > IBM® DB2 > Command Window**), then navigate to *DB2\_installdir\SQLLIB\java12* (where *DB2\_installdir* is the installation path of DB2).
  - b. Stop all DB2 services and make a note of the services you stopped, then run the following command to change your database installation to use JDBC 2.0: **usejdbc2.bat**
  - c. Restart the services that you stopped, then in a DB2 command window, issue the following command to ensure the essential DB2 processes are running: **db2start**.
3. Create the DB2 UDB SAMPLE database (if not already created) by completing the following steps:

- a. Select **Start > Programs > IBM DB2 > First Steps** .
- b. Click **Create Sample Databases**.
- c. Select the **DB2 UDB Sample** check box and click **OK**. (Note that if you created the sample database previously, the check box will be cleared.)
- d. When a message box indicates that the database has been created, click **OK** and close the First Steps wizard.
- e. If you would rather create a new database than use the DB2 UDB Sample database, open the DB2 command window and create a new database, for example **create database dreamauc**.

► Linux

1. If your database is not already installed, install it by running the installation program. Run the DB2 install program and select the **Create a DB2 instance** option, then accept the default user name db2inst1 and accept all remaining default values in the install program.
2. Login as user db2inst1 by issuing the following command: **su - db2inst1**
3. This step applies to DB2 7.2 only. DB2 8.1 does not require this step. Change your DB2 installation to use JDBC 2.0 by completing the following steps:
  - a. Navigate to the following directory: /home/db2inst1
  - b. Edit the file .bashrc and add the following lines of code to the bottom of the file:
 

```
if [ -f /home/db2inst1/sqllib/java12/usejdbc2 ]; then
    . /home/db2inst1/sqllib/java12/usejdbc2
fi
```
  - c. Save and close the file.
  - d. Issue the following command: **source ./bashrc**
4. Create the DB2 UDB SAMPLE database (if not already created) by completing the following steps:
  - a. Issue the following command: **db2sampl**
  - b. If you would rather create a new database than use the DB2 UDB Sample database, open the DB2 command window and create a new database.
5. Complete the following steps:
  - a. Exit from user db2inst1.
  - b. To ensure that the correct database libraries are accessible, complete the following steps:
    - 1) Navigate to the home directory. For example: /home/user\_name
    - 2) Edit the file .bashrc and add the following lines of code to the bottom of the file:
 

```
if [ -f /home/db2inst1/sqllib/db2profile ]; then
    . /home/db2inst1/sqllib/db2profile
fi
```

For DB2 7.2 only (not required for DB2 8.1):

```
if [ -f /home/db2inst1/sqllib/java12/usejdbc2 ]; then
    . /home/db2inst1/sqllib/java12/usejdbc2
fi
```
    - 3) Save and close the file.
    - 4) Issue the following command from the home directory: **source ./bashrc**
  - c. Exit from WebSphere Studio and then open WebSphere Studio again to pick up the environment changes.

## Create the database tables

Now you will run the mandatory database script, which creates several tables. To create a new project in WebSphere Studio that will hold the SQL file:

1. From the main menu, select **File > New > Project > Simple > Project**. Click **Next**.
2. In the **Project name** field, type AuctionSQL and click **Finish**.

To import the SQL file into the workspace:

1. Switch to the Data perspective (**Window > Open Perspective > Data**).
2. From the main menu, select **File > Import** to open the Import wizard, then select **File system** and click **Next**.
3. Beside the **From Directory** field, click **Browse** and then navigate to the following directory and select it (where *WSinstall\_dir* is the path where you have installed WebSphere Studio):  
*WSinstall\_dir*\samples\scenario\_parts\auction\v51
  - For DB2: crtOnlineItems\_db2.sql
  - For Cloudscape: crtOnlineItems\_cloudscape.sql
  - For iSeries: crtOnlineitems\_iSeries.sql
4. Click the **Browse** button beside the **Into Folder** field, select **AuctionSQL**, and click **Finish**.
5. In the Data Definition view, select the SQL file.
6. Right-click and select **Run on Database Server**. The Run Script wizard opens.
7. Clear the DROP statement checkboxes and click **Next** (This step is only required the first time you create the database).
8. Select the **Commit changes only upon success** radio button and click **Next**.
9. In the Database connection page of the wizard, enter the following values and click Finish:

Table 1. Database connection values

Field	DB2	Cloudscape	iSeries
Connection name	DB2	Cloudscape	iSeries
Database	sample	dreamauc	dreamauc
User ID	N/A	N/A	<i>your iSeries user ID</i>
Password	N/A	N/A	<i>your iSeries password</i>
Database vendor type	DB2 Universal Database	Cloudscape, V5.1	DB2 Universal Database for iSeries V5R1
JDBC driver	IBM DB2 APP DRIVER	Cloudscape Embedded JDBC Driver	AS/400® Toolbox for Java JDBC Driver
Host	N/A	N/A	<i>iSeries host</i>
Database location		<i>Cloudscape_dbdir</i> cloudscape\ dreamauc;create =true	N/A

Table 1. Database connection values (continued)

Field	DB2	Cloudscape	iSeries
Class location	<p>For Windows:  <i>DB2_installdir</i>/  java/db2java.zip</p> <p>For Linux:  /home/db2inst1/  sqllib/java12/  db2java.zip</p>	<i>WS_installdir</i> \runtimes\ base_v51\ lib\db2j.jar	jt400.jar

**Note:** *Cloudscape\_dbdir* refers to a directory on your file system where you want the Cloudscape database to be located. **create=true** in the Database location field is only required the first time you create the database.

For iSeries: The driver file that accesses the iSeries database must be locally available on your workstation. The driver file name is jt400.jar and is located in the IFS of the iSeries in /QIBM/ProdData/HTTP/Public/lib/jt400.jar. You can either map a network drive to this location or copy the jt400.jar file to a hard drive on your workstation. If you are running WebSphere Development Studio Client for iSeries, the jt400.jar file can also be found in *WSDC\_installdir*/java, where *WSDC\_installdir* is the directory where WebSphere Development Studio Client for iSeries is installed.

10. In the DB Servers view, confirm the table creation by right-clicking the connection and selecting **Refresh**. The required tables are created and populated with some test data.
11. For Cloudscape: You must disconnect Cloudscape before you make any more database connections or start the server, because Cloudscape is not shareable. To disconnect the Cloudscape database, right-click **Cloudscape** in the DB Servers view and select **Disconnect**.



---

## Chapter 3. Working with the Online Auction application

This topic tells you how to work with the Online Auction application by completing one or both of the following activities:

- Running the ready-made Online Auction sample application
- Building the Online Auction application for yourself

The sample application features an auction Web site, where you can browse auction items, submit items for sale, and bid on items. Running the application gives you a quick feel for the kind of J2EE application that you can build with WebSphere Studio. Also, the ready-made sample application contains a number of J2EE components that you can browse and reuse.

Building the application will give you a solid introduction to many of the WebSphere Studio tools, such as the EJB tools. To save time, some of the required components for building the application are already provided, such as the GIF images and some source code. The scenario is divided into modules, so that if you do not complete one module, you can skip ahead to the next module and load the completed components into your workspace. The self-contained modules are:

- Part 1 — Creating projects, CMP beans, and EJB queries
  - “Create the required projects” on page 12.
  - “Add CMP beans to the EJB project” on page 13.
  - “Adding EJB queries” on page 19.
  - “Generating deployment code” on page 22.
- Part 2 — Creating session beans and message-driven beans
  - “Creating session beans” on page 24.
  - “Add the business logic to the session bean” on page 26.
  - “Creating message-driven beans” on page 29
- Part 3 — Testing enterprise beans on a server, creating a Web service and running the application
  - “Create a server and add a data source” on page 32.
  - “Test the business methods” on page 35.
  - “Create the Web Service” on page 39
  - “Set up the team environment” on page 42.
  - “Version the application” on page 44.
  - “Add shared projects to another workspace” on page 44.
  - “Add a professional Web front end” on page 49.
  - “Debug the application” on page 51.


Even if you want to build the application, you may want to run the ready-made sample application first. This will show you how the application works before you actually build it, which will aid you in your development tasks.

---

## Running the ready-made sample application

**Note:** If you built the auction application *before* you followed these instructions on running the ready-made application, then you will need to remove the Auction project from the server before you can run the ready-made application. Instructions for removing the Auction project from the server are found in the final step of the “Test the application” section of Part 3.

To run the ready-made application:

1. Open WebSphere Studio.
2. Click the **Open The New Wizard** icon  at the left end of the toolbar:
3. In the left frame of the New wizard, expand **Examples** and select **Enterprise Applications 1.3**.
4. In the right frame, select **Auction** and click **Next**.
5. In the **Project name** field, accept the default name.
6. Click **Finish**. Instructions for running the Online Auction sample application are automatically opened in the Web browser.
7. Follow the instructions for running the sample application.
8. Repeat these steps to run the DreamCreditCard example.

---

## Part 1 — Creating projects, CMP beans, and EJB queries

### Create the required projects


In the workspace, everything must reside in a project. A project is the top level of organization of your resources in the workbench. A project contains files and folders. Projects are used for building, version management, sharing, and organizing resources. A project can contain session and persistent properties, settings for environmental variables, and references to other projects. As resources are created and modified within a project or projects, builders are run and constraints are maintained. Builders create or modify resources within projects, usually based on the existence and state of other resources. For example, a Java builder converts Java source files into executable class files, a Web link builder updates links to files when names and locations have changed, and so on.

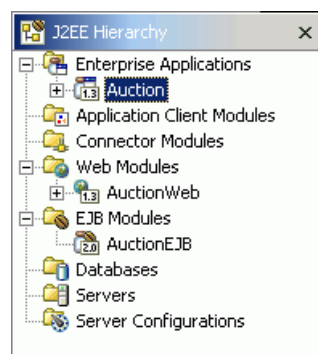
In this section, you create an Enterprise Application project named Auction. Enterprise Application projects contain the resources needed for enterprise applications and can contain a combination of Web modules, JAR files, EJB modules, and application client modules. An Enterprise Application project is deployed in the form of an EAR file. You also create an EJB project named AuctionEJB and a Web project named AuctionWeb, which are both referenced by the Auction enterprise application project. EJB projects contain the metadata files (such as the deployment descriptor, IBM extensions, and RDB mappings) for the EJB application, Java source files, compiled code for the enterprise beans, and stubs for the beans. An EJB project is deployed as an EJB module (JAR file). Web projects contain servlets, JSP files, Java files, static documents (for example HTML pages or images), and any associated metadata. A Web project is deployed as a Web module (WAR file). All three projects are created in a single step using a wizard.

**Note:** The next task assumes that the **Create EJB client JAR projects for new EJB projects** check box on the J2EE preferences page (**Window > Preferences > J2EE**) is cleared.



To create the three types of projects that are required for this application:

1. If you ran the ready-made sample application by following the instructions in “Running the ready-made sample application”, then you must remove the AuctionExample project from the server configuration by completing the following steps:
  - a. Switch to the Server perspective (**Window > Open Perspective > Other > Server**).
  - b. In the Server Configuration view, expand Servers and expand TestServer, then right-click AuctionExample and select **Remove**.
2. Switch to the J2EE perspective (**Window > Open Perspective > Other > J2EE**).
3. Click the **Open The New Wizard** icon  at the left end of the toolbar.
4. In the left frame of the New wizard, select **J2EE**.
5. In the right frame, select **Enterprise Application Project** and click **Next**.
6. Select the **Create J2EE 1.3 Enterprise Application project** radio button and click **Next**.
7. In the **Project name** field, type Auction.
8. (Optional) If you have enabled server targeting on the J2EE preferences page (**Window > Preferences > J2EE**), in the **Target Server** field, select **WebSphere Application Server v5.1**.
9. Click **Next**.
10. Click the **New Module** button. The New Module Project wizard opens.
11. Clear the **Application Client Project** and the **Connector Project** check boxes and click **Finish**.
12. Click **Finish** again.
13. In the J2EE Hierarchy view, expand Enterprise Applications, Web Modules, and EJB Modules to see the new projects. The Enterprise Application project named Auction, the EJB project named AuctionEJB, and the Web project named AuctionWeb are all added to the workspace:



## Add CMP beans to the EJB project

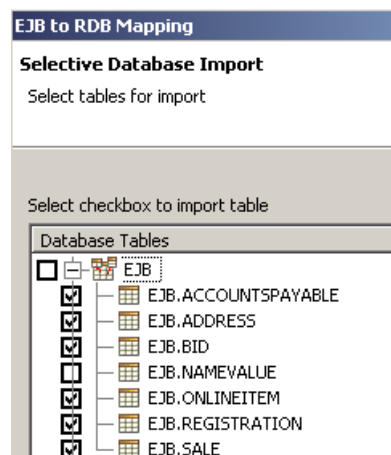
Entity beans with container-managed persistence (CMP) fields must be mapped to database tables that are represented in a schema. You map entity beans to database tables by creating a map. Maps are used to generate the SQL and other supporting code needed to make enterprise beans persistent. You will use the bottom-up mapping approach to create the CMP beans. This approach assumes that the database tables already exist, and that once the selected tables are imported, the enterprise beans and mappings between them are automatically generated. In this section, you learn how to add the following CMP beans to the AuctionEJB project:

- A CMP entity bean named Onlineitem (using the bottom-up mapping approach). The Onlineitem bean will store the item information on the Auction site.
- A CMP entity bean named Bid which will store the bidding history of each item
- A CMP entity bean named Accountspayable which will record the accounts payable information for users
- A CMP entity bean named Address which will store the addresses of the registered users
- A CMP entity bean named Registration which will store the registered users
- A CMP entity bean named Sale which will store the sale transaction information

You can also learn how to use the UML class diagram editor to create visualizations of the CMP beans that can assist with understanding the application, documentation, and collaboration.

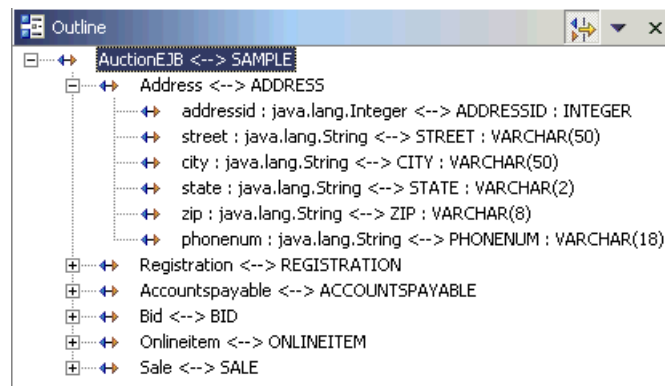
To add enterprise beans to the EJB project:

1. In the J2EE Hierarchy view, right-click AuctionEJB and select **Generate > EJB to RDB Mapping**. The EJB to RDB Mapping wizard opens.
2. Select **Create a new backend folder** and click **Next**.
3. Select **Bottom-up** and click **Next**. The Database Connection page of the wizard appears.
4. You can either use the connection you created earlier in the "Create the database tables" section by selecting the **Use existing connection** check box at the bottom of the wizard and selecting the connection, or you can create a new connection based on Table 1 in the previous topic. This instructs WebSphere Studio to read the database catalog for the dreamauc database and display a list of the tables in the database. Click **Next**.
5. Select the check boxes for the following tables and click **Next**:
  - EJB.ACCOUNTSPAYABLE
  - EJB.ADDRESS
  - EJB.BID
  - EJB.ONLINEITEM
  - EJB.REGISTRATION
  - EJB.SALE



6. Select the **Generate 2.0 enterprise beans** radio button.
7. In the **Package for generated EJB classes** field, type `com.acme.ejb`.

8. Click **Finish**. CMP entity beans named Accountspayable, Address, Bid, Onlineitem, Registration and Sale are created in the AuctionEJB project and the mapping editor opens.
9. Take a moment to look at the generated entity bean to table mapping. The Outline view shows a summary of how each bean in the AuctionEJB project is mapped to a table in the dreamauc database. For example, the Accountspayable entity bean is mapped to the ACCOUNTSPAYABLE table. In the Outline view, expand the Address mapping to see how the attributes of the Address bean map to columns in the Address table.



10. Alternatively, you can use the bottom pane of the mapping editor to see how bean attributes map to table columns. Under Enterprise beans column, expand **Address**. Each line represents an attribute to column mapping.

Overview	
Enterprise Beans	Tables
AuctionEJB	SAMPLE
Address	ADDRESS
addressid : java.lang.Integer	ADDRESSID : INTEGER
street : java.lang.String	STREET : VARCHAR(50)
city : java.lang.String	CITY : VARCHAR(50)
state : java.lang.String	STATE : VARCHAR(2)
zip : java.lang.String	ZIP : VARCHAR(8)
phonenumber : java.lang.String	PHONENUM : VARCHAR(18)
registration : Registration	FK_BILLINGADDRESS : ADDRESS
registration1 : Registration	FK_SHIPADDRESS : ADDRESS
Registration	REGISTRATION
Accountspayable	ACCOUNTSPAYABLE
Bid	BID
Onlineitem	ONLINEITEM
Sale	SALE

11. Close the mapping editor.

To disconnect the database connection:

1. In the DB servers view, right-click the connection and select **Disconnect**.

To add an extra ejbCreate() method to the Bid bean:

1. In the J2EE Hierarchy view, expand **AuctionEJB > Entity Beans > Bid** and double-click **BidBean**. The Java editor opens.
2. Add the following ejbCreate() method by typing or copying the following text in the editor:

```
/**
 * ejbCreate - Added to include the nullable fields
 */
```

```

public com.acme.ejb.BidKey ejbCreate(java.lang.Integer bidid,
RegistrationLocal registrationLocal,
OnlineitemLocal onlineItemLocal,
Long currentBid,
Long maxBid,
Integer bidInc )
throws javax.ejb.CreateException {
try {
    setBidid(bidid);
    setCurrentbid(currentBid);
    setMaximumbid(maxBid);
    setBidincrement(bidInc);
} catch (Exception e) {
    e.printStackTrace();
}

return null;
}

/**
 * ejbPostCreate to set the foreign keys
 */
public void ejbPostCreate(java.lang.Integer bidid,
RegistrationLocal registrationLocal,
OnlineitemLocal onlineItemLocal,
Long currentBid,
Long maxBid,
Integer bidInc )
throws javax.ejb.CreateException {
    setFk_bidderid(registrationLocal);
    setFk_itemtypeid(onlineItemLocal);
}

```

This `ejbCreate()` method will initialize all of the CMP fields, and this `ejbPostCreate()` method will initialize the container-managed relationship (CMR) fields. The CMR fields are used to manage relationships between beans. A container-managed relationship is an association that specifies relationships between enterprise beans and is represented by a CMR field in the deployment descriptor. These relationships are maintained by the EJB container. CMR fields cannot be modified in the `ejbCreate()` method; they must be set in the `ejbPostCreate()` method.

3. Save the file.
4. In the Outline view, right-click the new `ejbCreate()` method and select **Enterprise Bean > Promote to Local Home Interface**. The letter L appears to the left of the method names in the Outline view to indicate that they have been promoted to the local interface. When you use local interfaces instead of remote interfaces, the entity beans can only be accessed by clients running in the same JVM (Java Virtual Machine). Local interfaces reduce network traffic within an EJB container and increase performance of the enterprise bean.
5. Close the editor.

To modify the `SaleBean.java` file:

1. In the J2EE Hierarchy view, expand **AuctionEJB > Entity Beans > Sale** and double-click **SaleBean**. The Java editor opens.
2. Add the following `ejbCreate()` method by typing or copying the following text in the editor:

```

/**
 * ejbCreate method for a CMP entity bean.
 */
public com.acme.ejb.SaleKey ejbCreate(

```

```

        java.lang.Integer saleid,
        RegistrationLocal fk_sellerid, // missing after bottoms up
        OnlineitemLocal fk_itemid, // missing after bottoms up
        java.lang.Integer delivered,
        java.lang.Integer paid,
        java.sql.Timestamp datestarted,
        java.lang.String state)
        throws javax.ejb.CreateException {
        setSaleid(saleid);
        setDelivered(delivered);
        setPaid(paid);
        setDatestarted(datestarted);
        setState(state);
        return null;
    }
    /**
     * ejbPostCreate
     */
    public void ejbPostCreate(
        java.lang.Integer saleid,
        RegistrationLocal fk_sellerid, // missing after bottoms up
        OnlineitemLocal fk_itemid, // missing after bottoms up
        java.lang.Integer delivered,
        java.lang.Integer paid,
        java.sql.Timestamp datestarted,
        java.lang.String state)
        throws javax.ejb.CreateException {
        setFk_sellerid(fk_sellerid); // missing after bottoms up
        setFk_itemid(fk_itemid); // missing after bottoms up
    }
}

```

These foreign keys were missing from the methods after bottom-up EJB creation.

3. In the Outline view, right-click the new `ejbCreate()` method and select **Enterprise Bean > Promote to Local Home Interface**.
4. Save and close `SaleBean.java`.

## Generating getters

Now you need to generate getter methods on the key classes in the AuctionEJB project. Getters are methods that retrieve values for CMP fields. To generate getters:

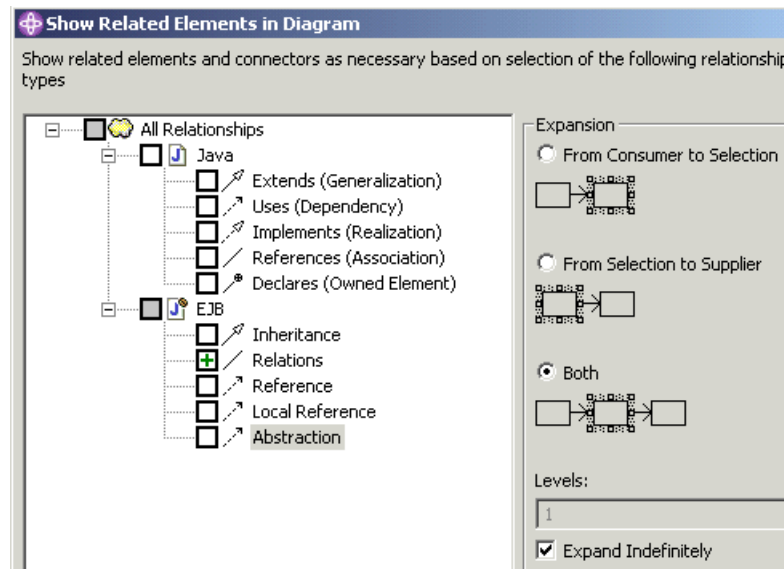
1. In the Project Navigator view, expand **AuctionEJB > ejbModule > com.acme.ejb > OnlineitemKey.java**. Double-click `OnlineitemKey.java` to open the file in the Java editor.
2. In the Outline view, right-click **itemtypeid : java.lang.Integer** and select **Source > Generate Getter and Setter**. The Generate Getter and Setter dialog box opens.
3. By default, both getter and setter methods will be selected for you. You only need the getter (`getItemtypeid()`) and not the setter. Clear the **setItemtypeid(Integer)** check box and click **OK**.
4. Save and close the editor.
5. In the Project Navigator view, expand **AuctionEJB > ejbModule > com.acme.ejb > RegistrationKey.java**. Double-click `RegistrationKey.java` to open the file in the Java editor.
6. In the Outline view, right-click **userid: java.lang.Integer** and select **Source > Generate Getter and Setter**.
7. Clear the **setUserId(Integer)** check box and click **OK**.
8. Save and close the editor.

## (Optional) Creating a UML class diagram visualization for the CMP beans



UML class diagram notation is useful for creating an abstract view of the entity beans, showing the entity bean CMP fields and relationships between the beans. In this section, you will learn how to create a class diagram, populate and expand the diagram, and create a filtered view showing just the beans, with their CMP fields and CMR relationships.

To create a new diagram:

1. In the Project Navigator view, right-click the AuctionEJB folder and click **New > Other**. The New wizard opens.
2. In the left frame, select **Simple**.
3. In the right frame, select **Folder** and click **Next**.
4. In the **Folder name** field, type **diagrams**, and then click **Finish**.
5. Click **File > New > Class Diagram**. The New Class Diagram wizard opens.
6. Select or type **AuctionEJB/diagrams** as the parent folder.
7. In the **File name** field, type **AuctionBeans** and click **Finish**. The Class Diagram editor opens.
8. In the J2EE Hierarchy view, expand **AuctionEJB > Entity Beans**. Select the **Onlineitem** bean and drag it onto the Class Diagram Editor window. A UML view of the **Onlineitem** bean is created in the Class Diagram editor.
9. Right-click the **Onlineitem** view in the diagram and select **Show Related Elements**. The Show Related Elements in Diagram dialog box opens.
10. Clear the **Java** check box and clear the **Inheritance**, **Reference**, **Local Reference**, and **Abstraction** check boxes.
11. Select **Expand Indefinitely**, and then click **Yes** when prompted. The Show Related Elements in Diagram dialog box opens:



12. Click **OK**. The diagram is expanded with the **Sale**, **Address**, **Registration**, **Bid**, and **AccountsPayable** beans.
13. Right-click the diagram and select **Select > Select All Shapes**. The beans on the diagram are highlighted.

14. From the Class Diagram editor toolbar, click the drop-down menu for the **Show/Hide Compartment** button , and then clear Operation Compartment. The operations compartment of each bean view is hidden.
15. From the toolbar, click the **Arrange Elements** button . The diagram is automatically rearranged.
16. (Optional) Drag the Sale bean on top of the Onlineitem bean and to the left of the Address bean. Drag the connectors representing the CMR relationships Sale to Registration and Sale\_to\_Registration4 to minimize the label overlap. Do the same for the connectors representing the Registration\_to\_Address and Registration\_to\_Address3 CMR relationships.
17. Save the diagram. The diagram appearance should be similar to the following:

18. Close the diagram.

The EJB Query Language (EJB QL) is a standard and portable SQL-like language used for querying entity beans. Its syntax allows for the definition of the query in terms of the object model, making it independent of the database used to persist the beans. You can add queries to the beans using the deployment descriptor editor or the UML class diagram editor. If you want to use the UML class diagram editor, skip to the *Adding EJB queries using the UML class diagram editor* section below. To add queries using the deployment descriptor editor:



1. In the J2EE Hierarchy view, right-click **AuctionEJB** and select **Open With > Deployment Descriptor Editor**.
2. Click the **Beans** tab, and select **Bid**.
3. Scroll down to the Queries section and click **Add**. The Add Finder Descriptor wizard opens.
4. Click the **New** radio button for the method.
5. Click the **find method** radio button for the method type.
6. Ensure the **Local** radio button is selected.
7. In the **Name** field, type `findAutoBids`.
8. Click the **Add** button beside the Parameters list box. The Add Method Parameter dialog box opens.
9. In the **Name** field, type `item`.
10. In the **Type** field, select `java.lang.Integer` and click **OK**.
11. In the **Return type** field, select `java.util.Collection` and click **Next**.
12. Type the following string into the **Query statement** text area and click **Finish**:  

```
select object(b) from Bid b where b.fk_itemtypeid.itemtypeid =?1 and
b.maximumbid >=b.fk_itemtypeid.lastbid + b.bidincrement
```

The Queries section now shows the query that we just created for the Onlineitem bean.

13. To take a look at the actual XML elements that were added to the deployment descriptor, click the **Source** tab at the bottom of the editor. The deployment descriptor source is displayed in the editor.
14. In the Outline view, expand the Bid bean and select the query. The corresponding lines of code are highlighted in the editor.  

```
<query>
<description></description>
<query-method>
  <method-name>findAutoBids</method-name>
  <method-params>
    <method-param>java.lang.Integer</method-param>
  </method-param>
</query-method>
<ejb-ql>select object(b) from Bid b where b.fk_itemtypeid.itemtypeid = ?1 and
b.maximumbid >= b.fk_itemtypeid.lastbid + b.bidincrement</ejb-ql>
</query>
```
15. Create four more queries for the following beans following the steps described above:

*Table 2. Values for creating queries*

Bean	Bid	Onlineitem	Registration	Sale
Method	New	New	New	New
Method Type	find method	find method	find method	find method
Type	Local	Local	Local	Local
Name	findHighestBid	findByValue	findAll	findExpired Items
Parameter Name	item	value	N/A	currentTS
Parameter Type	java.lang.Integer	long	N/A	java.sql. Timestamp
Return type	com.acme.ejb. BidLocal	java.util. Collection	java.util. Collection	java.util. Collection



Table 2. Values for creating queries (continued)

Bean	Bid	Onlineitem	Registration	Sale
Query statement	select object (b) from Bid b where b. fk_itemtypeid. itemtypeid=?1 and b.currentbid= (select max (b1.currentbid) from Bid b1 where b1. fk_itemtypeid. itemtypeid =?1)	select object (o) from Onlineitem as o where o.value <=?1	select object (r) from Registration r	select distinct object (o) from Sale as o where (o.fk_buyerid is null) and (o.fk_itemid. endbidding<= ?1) and (o.fk_itemid. lastbid is not null)

The findHighestBid query returns only one record from the database (by specifying com.acme.ejb.BidLocal as the Return type), which is the highest bid. For the findByValue query for the Onlineitem bean, value is the name of a column in the Onlineitem table and ?1 represents the parameter that is passed into the finder method, so the string specifies that you want all of the items in the auction that are less than or equal to the amount that is passed in. The findAll query for the Registration bean finds all registered users. The findExpiredItems query for the Sale finds items that have not been bought when the bidding time has ended.

- Click **File > Save** to save the changes, then close the deployment descriptor editor.

### (Optional) Adding EJB queries using the UML class diagram editor

You can use the UML Class Diagram editor to edit beans. In this section, you can use the Class Diagram editor instead of the deployment descriptor editor to add a query to a bean.

If you did not complete the optional steps in the *Creating a UML class diagram visualization for the CMP beans* section, you will need to create a folder where we will store the diagrams. If you completed the optional section, then the folder is already created. To create a folder for the diagrams:

- In the Project Navigator view, right-click the AuctionEJB folder and click **New > Folder**. The New folder wizard opens.
- In the **Folder name** field, type diagrams, and click **Finish**.

To create a new class diagram:

- Click **File > New > Class Diagram**. The New Class Diagram wizard opens.
- Select or type AuctionEJB/diagrams as the parent folder.
- In the **File name** field, type AuctionQueries. The Class Diagram editor opens.
- In the J2EE Hierarchy view, expand **AuctionEJB > Entity Beans**.
- Select the Bid, Onlineitem, Registration, and Sale beans.
- Drag the beans onto the Class Diagram editor. The UML views for each of the beans are created.
- Right-click the class diagram and select **Show/Hide Relationships**. The Show/Hide Relationships dialog box opens.
- Click **All Relationships** until the hide ☐ option is selected, and then click **OK**. The connectors representing the CMR relationships are removed from the diagram.

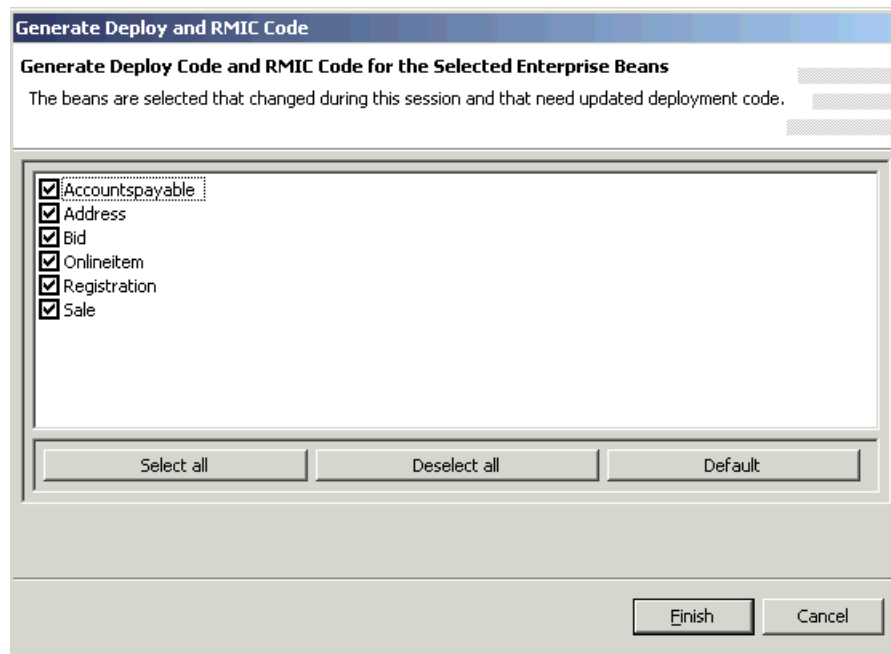
9. In the Class Diagram editor, right-click the Bid view, and select **Add EJB > Query**. The Add Finder Descriptor wizard opens.
10. Click **New** for the method.
11. Click **Find method** for the method type.
12. Ensure **Local** is selected.
13. In the **Name** field, type findAutoBids.
14. Click the **Add** button beside the Parameters list box. The Add Method Parameter dialog box opens.
15. In the **Name** field, type item.
16. In the **Type** field, select **java.lang.Integer** and click **OK**.
17. In the **Return type** field, select **java.util.Collection** and click **Next**.
18. Type the following string into the **Query statement** text area :  

```
select object(b) from Bid b where b.fk_itemtypeid.itemtypeid=?1 and
b.maximumbid >=b.fk_itemtypeid.lastbid+ b.bidincrement
```
19. Click **Finish**. The operations compartment for the Bid bean has been automatically updated and now shows the finder method, findAutoBids, that you just created.
20. Create four more queries for the following beans following the steps described above. Use the values above in the *Values for creating queries* table.
21. Click **File > Save** to save the changes, and then close the Class Diagram editor.

## Generating deployment code

To generate deployment code for the enterprise beans:

1. Right-click **AuctionEJB** and select **Generate > Deployment and RMIC code**.
2. Click **Select all** to mark all of the beans for code generation:

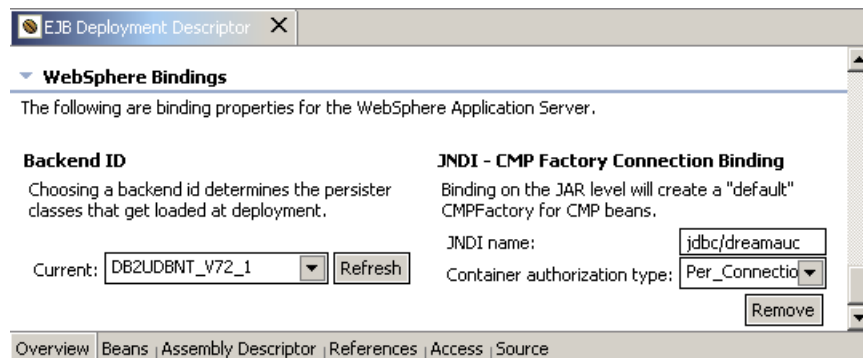


3. Click **Finish** to generate the deployment and RMIC code. This code is needed to deploy the EJB module to a server.

4. After the code has been generated, expand AuctionEJB and the ejbModule folder in the Project Navigator to see the packages and classes that were created:
  - com.acme.ejb — The concrete classes.
  - com.acme.ejb.websphere\_deploy — This package contains common classes used to support persisting entity beans running in WebSphere regardless of the data source.
  - com.acme.ejb.websphere\_deploy.Database\_type — This package contains classes that are specific to persisting entity beans to the database you are using, for example, com.acme.ejb.websphere\_deploy.CLOUDSCAPE\_V50\_1.

To configure the EJB module to use the correct JNDI name for the auction database:

1. In the J2EE Hierarchy view, right-click **AuctionEJB**.
2. Select **Open With > Deployment Descriptor Editor**.
3. Scroll down to the WebSphere Bindings section. This section shows the type of backend used for persisting data in the project and the data source that will be used.
4. In the **JNDI name** field, type jdbc/dreamauc for the JNDI name. This setting applies to all EJB 2.0 CMP beans in the project. Later, the JNDI name will be paired up with a JNDI name in the server configuration, so that a connection to the real database can be made.
5. Now you must specify the Container authorization type to be used when connecting to the database. In the **Container authorization type** field, select **Per\_Connection\_Factory**. This setting uses the data source properties for authorization.



6. Click **File > Save** and then close the deployment descriptor editor.

## Part 2 — Creating session beans and message-driven beans

Now you need to create an EJB project that will hold the business logic for the application. If you have not completed Part 1, you can import AuctionEJB.jar into your workspace. Otherwise, proceed to the Creating session beans section.

**Note:** If you completed some of the tasks in Part 1, you must delete the AuctionEJB project from your workspace. In the J2EE Hierarchy view, right-click **AuctionEJB** and select **Delete**. Click **OK**. Select **Also delete contents under file\_system\_location**, where *file\_system\_location* is the directory where the project is stored on the file system.

(Optional) The contents of the AuctionEJB project have been provided for you in a JAR file. To import the JAR file:

1. From the **File** menu, select **Import** to open the Import wizard, then select **EJB JAR file** and click **Next**.
2. Beside the **EJB JAR file** field, click **Browse** and then navigate to the following file and select it (where *WS\_installdir* is the directory where you have installed WebSphere Studio):

Windows

*WS\_installdir*\samples\scenario\_parts\auction\v51\AuctionEJB.jar

> Linux

*WS\_installdir*/samples/scenario\_parts/auction/v51/AuctionEJB.jar

Click **Open**.

3. In the **Project** field, type AuctionEJB.
4. In the **EAR project** field, type Auction.
5. Click **Finish**. Click **Yes** if prompted. The files contained in AuctionEJB.jar are imported into the AuctionEJB project.
6. To correct any validation errors, right-click **Auction** and select **Run Validation**. The errors should disappear from the Tasks list.
7. In the J2EE Hierarchy view, right-click **AuctionEJB**.
8. Select **Open With > Deployment Descriptor Editor**.
9. Scroll down to the WebSphere Bindings section. In the **Current** field, ensure the backend ID is the database that you are working with. For example, if you are working with DB2 7.2 , select **DB2UDBNT\_V72\_1**, or for DB2 8.1, select **DB2UDBNT\_V8\_1**.
10. Generate deployment code for the AuctionEJB project by right-clicking it in the J2EE Hierarchy view and selecting **Generate > Deployment and RMIC code**.
11. Click **Select all** to mark all of the beans for code generation and click **Finish** to generate the deployment and RMIC code.

## Creating session beans

In this module, we will create a new EJB project called AuctionEJBBiz to contain the session beans and a message-driven bean. We are storing the session beans and message-driven beans in a separate EJB project from the CMP beans. This is a recommended practice because during the course of development, you may want to change the database schema. When you change the database schema, it is easier to delete the project containing the CMP beans and regenerate the CMP beans from a new bottom-up mapping than to modify the existing CMP beans. If you had stored the session beans in the same EJB project as the CMP beans, they would be lost and you would have to manually recreate them again. By keeping the CMP beans in a separate EJB project, you could safely delete that project without losing any other enterprise beans.

The partial contents of the AuctionEJBBiz project have been provided for you in a JAR file. To import the JAR file:

1. From the **File** menu, select **Import** to open the Import wizard, then select **EJB JAR file** and click **Next**.
2. Beside the **EJB JAR file** field, click **Browse** and then navigate to the following file and select it (where *WS\_installdir* is the directory where you have installed WebSphere Studio):

Windows

`WS_installdir\samples\scenario_parts\auction\v51\AuctionEJBizPartial.jar`


Linux

`WS_installdir/samples/scenario_parts/auction/v51/AuctionEJBizPartial.jar`

Click **Open**.

3. Beside the **Project** field, click **New**. The New EJB Project wizard opens.
4. In the **Project name** field, type `AuctionEJBiz`.
5. (Optional) If you have enabled server targeting, in the **Target Server** field, select **WebSphere Application Server v5.1**.
6. In the **EAR project** field, select `Auction` and click **Next**.
7. Click **Finish**. If you are prompted to overwrite the `META-INF/MANIFEST.MF` file, click **Yes**. The files contained in `AuctionEJBizPartial.jar` are imported into the newly created `AuctionEJBiz` project. Any errors that appear in the Tasks list will be resolved in subsequent steps.
8. To set the module dependency for the `AuctionEJBiz` project to the `AuctionEJB` project, right-click the `AuctionEJBiz` project and select **Properties**. The Properties page opens.
9. On the left frame of the Properties page, click **Java JAR Dependencies**.
10. In the Available dependent JARs list table, click the **AuctionEJB.jar** check box and click **OK**. This enables the `AuctionEJBiz` project to access the classes in the `AuctionEJB` project at run time.

Now you will create a stateless session bean named `ItemHelper` (using the Create an Enterprise bean wizard). Session beans are non-persistent enterprise beans that do not require database access, though they can obtain it indirectly (as needed) by accessing entity beans. To create a session bean in the `AuctionEJBiz` project:

1. In the J2EE Hierarchy view, select **AuctionEJBiz** and click the **Create an Enterprise Bean** icon . The Enterprise Bean Creation wizard opens.
2. In the **EJB Project** field, select **AuctionEJBiz** and click **Next**.
3. Ensure that the **Session bean** radio button is selected.
4. In the **Bean name** field, type `ItemHelper`. The bean name will be used as a prefix for the names of the generated class files.
5. In the **Default package** field, type `com.acme.ejb.biz` and click **Next**.
6. Select the **Stateless** radio button as the Session type. Ensure that the **Container** radio button is selected as the Transaction type.
7. Ensure that the **Local client view** checkbox is selected and that the **Remote client view** checkbox is cleared. Local client views help to reduce network traffic and other overhead from calls between enterprise beans in the same container.
8. Ensure the fields on the Enterprise Bean Details page are filled in with the following values:
  - **Bean supertype:** `<none>`
  - **Bean class:** `com.acme.ejb.biz.ItemHelperBean`
  - **EJB binding name:** `ejb/com/acme/ejb/biz/ItemHelperLocalHome`
  - **Local home interface:** `com.acme.ejb.biz.ItemHelperLocalHome`
  - **Local interface:** `com.acme.ejb.biz.ItemHelperLocal`

9. Click **Finish**. A session bean named `ItemHelper` is created in the `AuctionEJB` project, which will be used to hold the application's business logic and serve as a front end for the `Onlineitem` CMP entity bean.

## Add the business logic to the session bean

In this section, you add your business logic to the session bean and then promote the bean to the local interface. The local interface defines the business methods that can be called by a client. The business logic consists of four public methods:

- **createItem** Registers a new item for sale in the auction. The method ensures that the business rules are enforced, such as ensuring that the value of the minimum opening bid is less than the appraised value of the item.
- **findByValue** Searches the auction database for all items that have a value less than the value specified as the search criteria.
- **findItemByKey** Wraps the `findByPrimaryKey` method of the `Onlineitem` bean and returns a data transfer object
- **processExpiredItems** Processes the credit card payment using the Web service.

To save you the effort of typing in the business methods by hand, the business methods are provided for you in a `.java` file that you will import into the `AuctionEJB` project. To import the `ItemHelperBean.java` file:

1. From the **File** menu, select **Import** to open the Import wizard, then select **File System** and click **Next**.
2. Beside the **From Directory** field, click **Browse** and then navigate to the following directory (where `WS_installdir` is the directory where you have installed WebSphere Studio):

Windows

`WS_installdir\samples\scenario_parts\auction\v51`

Linux

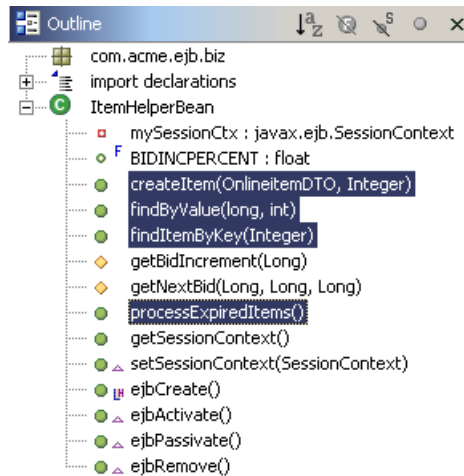
`WS_installdir/samples/scenario_parts/auction/v51`

Click **OK**.

3. Select the **ItemHelperBean.java** checkbox.
4. Beside the **Into Folder** field, click **Browse** and select the **AuctionEJB/ejbModule/com/acme/ejb/biz** folder, then click **OK**.
5. Click **Finish**. If prompted to overwrite the existing file, click **Yes**. The `ItemHelperBean.java` is imported into the `AuctionEJB` project.

To promote business methods to local interfaces:

1. In the J2EE Hierarchy view, expand **EJB Modules**, **AuctionEJB**, **Session Beans** and **ItemHelper**, then double-click **ItemHelperBean**. The class opens in the Java editor.
2. In the Outline view, expand **ItemHelperBean** and select the **createItem**, **findByValue**, **findItemByKey**, and **processExpiredItems** methods together:



3. Right-click the methods and select **Enterprise Bean > Promote to Local Interface**.

Notice the L label decoration appears beside the method names in the Outline view to indicate that these are local methods.

(Optional) You can browse the source code for these four methods to give you an idea of how to add business logic to an application.

## Adding EJB references to enterprise beans

An EJB reference is a logical name used to locate the home interface of an enterprise bean used by an application. The session beans in this application use logical names under the `java:comp/env/ejb` JNDI naming context to reference other enterprise beans. J2EE applications deployed in the WebSphere Application Server environment should use the `java:comp/env/ejb` naming context to look up enterprise beans instead of using the JNDI name. This approach avoids hard coding of the target JNDI name in the Java files which would require recompilation any time you needed to make a change. Instead, the EJB references are specified in the deployment descriptor and no recompilation of code is needed. With this naming context, naming conflicts will be avoided and applications will be much more portable.

Before you can run the enterprise beans on a server, you must bind the references to the target server. For example, the ItemHelper session bean needs to call the Onlineitem CMP bean in the `findByValue` (long value, int currency) method. It uses the ServerHelper.java class in the `com.acme.common` package to look up the logical name of the Onlineitem bean which is `java:comp/env/ejb/Onlineitem`. You need to define an EJB local reference called `ejb/OnlineItem` for the ItemHelperBean and bind it to the JNDI name of the Onlineitem bean in the server which is `ejb/com/acme/OnlineitemLocalHome`.

## Adding references to the ItemHelper bean

You can add references to the ItemHelper bean using the deployment descriptor editor or the UML class diagram editor. If you want to add references using the UML class diagram editor, skip to the *Adding references to the ItemHelper bean using the UML class diagram editor* section below. To add references to the ItemHelper bean using the deployment descriptor editor:



1. In the J2EE Hierarchy view, right-click **AuctionEJBBiz** and select **Open With > Deployment Descriptor Editor**. The editor opens.
2. Click the **References** tab.
3. Select ItemHelper and click **Add**. The Add Reference wizard opens.

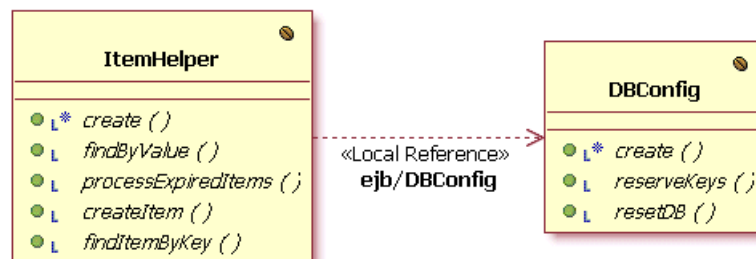


4. Select the **EJB local reference** radio button and click **Next**.
5. Select the **Enterprise bean in current EJB project** radio button.
6. Select **DBConfig**. The fields describing the local reference to the Profile bean are now populated in the Add EJB Local Reference wizard.
7. Click **Finish**.
8. Click the **Add** button on the References page again.
9. Select the **EJB local reference** radio button and click **Next**.
10. Select the **Enterprise bean in different EJB project** radio button. Expand **Auction** and **AuctionEJB** and select **Sale**.
11. Click **Finish**. Add three more EJB local references from the AuctionEJB project called `ejb/Bid`, `ejb/Registration` and `ejb/Onlineitem`, using the steps described above.
12. Save and close the editor.

### (Optional) Adding references to the ItemHelper bean using the UML class diagram editor

To add references to the ItemHelper bean using the UML class diagram editor:

1. In the Project Navigator view, right-click the AuctionEJBBiz project, and then select **New > Other**. The New folder wizard opens.
2. In the left frame, select **Simple**.
3. In the right frame, select **Folder** and click **Next**.
4. In the **Folder name** field, type `diagram` and click **Finish**.
5. Click **File > New > Class Diagram**. The New Class Diagram wizard opens.
6. Select or type `AuctionEJBBiz/diagrams` as the parent folder, and in the **File name** field, type `References` and click **Finish**. The Class Diagram editor opens.
7. In the J2EE Hierarchy view, expand **AuctionEJBBiz > Session Beans**, and then select the `ItemHelper` bean and drag the `ItemHelper` bean onto the Class Diagram editor. A UML view for `ItemHelper` is created.
8. Drag the `DBConfig` bean from the J2EE Hierarchy view to the class diagram editor.
9. From the Class Diagram editor EJB 2.0 tool palette , select the **Create an EJB Local Reference** tool .
10. In the Class Diagram editor, click the `ItemHelper` bean and drag the connector to the `DBConfig` bean, and then release the mouse button. An EJB local reference relationship is created between the `ItemHelper` and `DBConfig` beans:



11. Click **File > Save** to save the diagram.
12. Double-click the `ItemHelper` view. The deployment descriptor editor opens. Close the class diagram editor.
13. Click the **References** tab.




14. Select ItemHelper and click **Add**. The Add Reference wizard opens.
15. Select the **EJB local reference** radio button and click **Next**.
16. Click **Enterprise bean in different EJB project**, and then expand **Auction** and **AuctionEJB** and select **Sale**.
17. Click **Finish**.
18. Add three more EJB local references from the AuctionEJB project called ejb/Bid, ejb/Registration and ejb/Onlineitem, using the steps described above.
19. Save and close the editors.

## Creating message-driven beans

Message-driven beans (MDB) are the new type of enterprise beans introduced in the EJB 2.0 specification. They are server-side components that are designed to receive asynchronous messages sent by Java Message Service (JMS) clients. JMS allows applications to create, send, receive, and read messages. When a client sends a message to a JMS destination (queue), the container automatically notifies any MDB listening to the port associated with the destination. The MDB can then process the message as required by the application. You will develop a MDB for the Auction application that receives bids and bids on the user's behalf if they have enabled auto-bidding. When a message is received in the queue associated with the bean's listener port, the container invokes the bean's `onMessage(Message)` method.

To create a message-driven bean in the AuctionEJBBiz project:

1. In the J2EE Hierarchy view, select **AuctionEJBBiz** and click the **Create an Enterprise Bean** icon . The Enterprise Bean Creation wizard opens.
2. In the **EJB Project** field, select **AuctionEJBBiz** and click **Next**.
3. Ensure that the **Message-driven bean** radio button is selected.
4. In the **Bean name** field, type BidProxy. The bean name will be used as a prefix for the name of the generated bean class.
5. In the **Default package** field, type `com.acme.ejb.biz` and click **Next**.
6. Ensure that the **Container** radio button is selected as the Transaction type.
7. Ensure the fields on the Enterprise Bean Details page are filled in with the following values:
  - **Destination Type:** Queue
  - **Bean Supertype:** <none>
  - **Bean class:** `com.acme.ejb.biz.BidProxyBean`
  - **ListenerPort name:** P2PLListenerPort
8. Click **Finish**. A message-driven bean named BidProxy is created in the AuctionEJBBiz project, which will be used to receive messages from the BidHelper bean.
9. (Optional) To see the values you specified in the wizard, right-click the AuctionEJBBiz project in the J2EE Hierarchy view and select **Open With > Deployment Descriptor Editor**. On the Beans page, select **BidProxy** and view the associated values in the editor.

The contents of the message-driven bean are provided for you in the BidProxyBean.java file which you must import into the AuctionEJBBiz project:

1. From the **File** menu, select **Import** to open the Import wizard, then select **File System** and click **Next**.

2. Beside the **From Directory** field, click **Browse** and then navigate to the following directory and select the **BidProxyBean.java** checkbox (where *WS\_install\_dir* is the directory where you have installed WebSphere Studio):

Windows

*WS\_install\_dir*\samples\scenario\_parts\auction\v51

Linux

*WS\_install\_dir*/samples/scenario\_parts/auction/v51

Click **Open**.

3. Beside the **Into Folder** field, click **Browse** and select the **AuctionEJBBiz/ejbModule/com/acme/ejb/biz** folder, then click **OK**.
4. Click **Finish**. If prompted to overwrite the existing file, click **Yes**. The **BidProxyBean.java** is imported into the **AuctionEJBBiz** project.

To add references to the **BidProxy** bean:

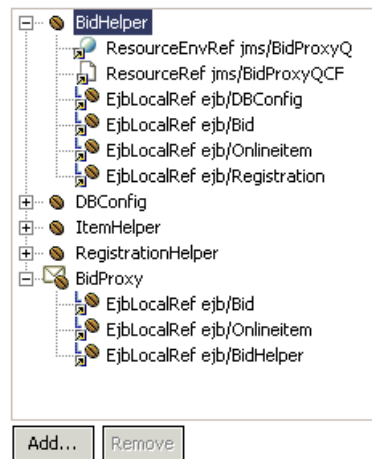
1. In the J2EE Hierarchy view, right-click **AuctionEJBBiz** and select **Open With > Deployment Descriptor Editor**. The editor opens.
2. Click the **References** tab.
3. Select **BidProxy** and click **Add**. The Add Reference wizard opens.
4. Add two EJB local references from the **AuctionEJB** project called **ejb/Bid** and **ejb/Onlineitem**.
5. Add an EJB local reference from the **AuctionEJBBiz** project called **ejb/BidHelper**.
6. Save the editor.

Now you need to add references to the **BidHelper** session bean. Since the **BidHelper** session bean will send JMS messages, resource environment references and resource references are required. The method **BidHelper.bidItem()** sends a JMS message to inform the **BidProxy** message-driven bean when someone has bid on an item. The **BidHelper** session bean must access the queue connection factory and the queue from the JMS server that you will configure later in the scenario. You must set up an EJB resource reference for the queue connection factory and an EJB resource environment reference for the queue. To add these references:

1. On the **References** page of the EJB deployment descriptor editor, select **BidHelper** and click **Add**. The Add Reference wizard opens.
2. Select the **Resource environment reference** radio button and click **Next**.
3. In the **Name** field, type **jms/BidProxyQ**.
4. In the **Type** field, type **javax.jms.Queue** and click **Finish**. The Type field indicates the type of connection factory for the JMS server.
5. Expand **BidHelper** and select the reference you just created, **ResourceEnvRef jms/BidProxyQ**.
6. Under the WebSphere Bindings section, in the **JNDI name** field, type **jms/bidProxyQ**.
7. Select **BidHelper**.
8. Click the **Add** button on the References page again.
9. Select the **Resource reference** radio button and click **Next**. Resource references are needed so that different projects may access common resources. The resource is a connection factory that is bound to a JNDI name that the server will use at run time.
10. In the **Name** field, type **jms/BidProxyQCF**.

11. In the **Type** field, select `javax.jms.QueueConnectionFactory`.
12. In the **Authentication** field, select **Application**.
13. In the **Sharing scope** field, select **Shareable**. Connections can be shareable or unshareable.
14. Click **Finish**.
15. Select the **ResourceRef** `jms/BidProxyQCF` reference.
16. Under the WebSphere Bindings section, in the **JNDI name** field, type `jms/bidProxyQCF`.
17. There should now be six references under the BidHelper bean on the References page.

## References



Save and close the editor.

To generate deployment code for the enterprise beans:

1. Right-click **AuctionEJBBiz** and select **Generate > Deployment and RMIC code**.
2. Select all of the beans except BidProxy. You do not need to generate deployment code for the BidProxy bean because it is a message-driven bean and does not have any interfaces that a client would need to access.
3. Click **Finish** to generate the deployment and RMIC code.

## Part 3 — Testing enterprise beans on a server, creating a Web service and running the application

If you have not completed Part 2, you can import the completed AuctionEJBBiz.jar from the following location:

1. From the **File** menu, select **Import** to open the Import wizard, then select **EJB JAR file** and click **Next**.
2. Beside the **JAR file** field, click **Browse** and then navigate to the following file and select it (where `WS_installdir` is the directory where you have installed WebSphere Studio):

Windows

`WS_installdir\samples\scenario_parts\auction\v51\AuctionEJBBiz.jar`



Linux

`WS_installdir/samples/scenario_parts/auction/v51/AuctionEJBBiz.jar`

3. Beside the **Project** field, click **New**. The New EJB Project wizard opens.
4. In the **Project name** field, type AuctionEJBiz.
5. In the **EAR project** field, select Auction and click **Next**.
6. Click **Finish**. If you are prompted to overwrite the META-INF/MANIFEST.MF file, click **Yes**. The files contained in AuctionEJBiz.jar are imported into the AuctionEJBiz project.
7. Right-click the AuctionEJBiz project and select **Properties**.
8. On the left frame of the Properties page, click **Java JAR Dependencies**.
9. In the Available dependent JARs list table, click the **AuctionEJB.jar** check box and click **OK**. This enables the AuctionEJBiz project to access the classes in the AuctionEJB project at run time.
10. Generate deployment code for the AuctionEJBiz project by right-clicking it in the J2EE Hierarchy view and selecting **Generate >Deployment and RMIC code**.
11. Select all of the beans except BidProxy and click **Finish** to generate the deployment and RMIC code.

## Create a server and add a data source



To test the enterprise beans, you need to create a server. To create and configure a server:

1. Right-click the Servers folder in the J2EE Hierarchy view. Select **New > Server and Server Configuration**. The Create a New Server and Server Configuration wizard opens.
2. In the **Server Name** field, type TestServer.
3. In the **Folder** field, select **Servers** from the drop-down menu.
4. In the **Server type** list box, expand **WebSphere version 5.1** and select **Test Environment**.
5. Click **Finish** to create the server.
6.  For DB2:
  - a. In the J2EE Hierarchy, expand **Servers**.
  - b. Double-click **TestServer**. The server editor opens.
  - c. In the server editor, click the **Environment** tab, then click **Add External JARs** in the **WebSphere specific class path (ws.ext.dirs)** section to add **DB2\_installdir/java12/db2java.zip** to the WebSphere path, where **DB2\_installdir** is the directory where DB2 is installed.  
 For iSeries: In the server editor, click the **Environment** tab, then click **Add External JARs** in the **WebSphere specific class path (ws.ext.dirs)** section to add the jt400.jar file to the path.
  - d.  Click **File > Save (name of file)** to save your changes.

Now you need to add a data source so that the server can access the data in the dreamauc database. You will need to complete the following tasks:


- Defining a JAAS authentication entry
- Defining a JDBC provider
- Defining a data source
- Defining resource properties for the data source, such as userids and passwords

To define a JAAS authentication entry (skip this section if you are using Cloudscape):

1. In the J2EE Hierarchy view, expand **Servers**.
2. Double-click **TestServer**. The server editor opens.
3. At the bottom of the editor, click the **Security** tab.
4. In the JAAS Authentication Entries section, click **Add**. The JAAS Authentication Entry dialog opens.
5. In the Alias field, type your database user ID.  For Linux this is db2inst1 for a DB2 database.  
For iSeries: Type the name of the iSeries server.
6. In the User ID field, type your database user ID again.  For Linux this is db2inst1 for a DB2 database.  
For iSeries: Type your iSeries user ID.
7. In the Password field, type your database password then click **OK**.  
For iSeries: Type your iSeries password.

To create a JDBC provider:

1. At the bottom of the editor, click the **Data source** tab.
2. In the Server Settings section, beside the JDBC provider list table, click **Add**. The Create a JDBC Provider wizard opens.
3. In the **Database type** list box, select **IBM DB2** or **Cloudscape**.
4. In the **JDBC provider type** list box, select one of the following provider types and click **Next**:
  - For Cloudscape: **Cloudscape JDBC Provider (XA)** You can safely ignore the description stating that this provider is deprecated.
  - For DB2: **DB2 Legacy CLI-based Type 2 JDBC Driver (XA)**
  - For iSeries: **DB2 UDB for iSeries (Toolbox XA)**

The XA provider supports the two-phase commit protocol, which is when multiple changes across multiple resources such as beans can be treated as a single transaction. This is required by the BidHelper session bean as it will update the database via the CMP beans and also send a JMS message in a single transaction.
5. In the **Name** field, type a name for your JDBC provider, for example, DB2 JDBC Type 2 Provider (XA) or iSeries Toolbox XA.
6. For iSeries: Click the **Remove** button beside the **Class path** list box to delete the default entry. Beside the Class path list box, click **Add External JARs** and browse to the **jt400.jar**.
7.  For DB2: For DB2: Click the **Remove** button beside the **Class path** list box to delete the default entry. Beside the Class path list box, click **Add External JARs** and browse to the DB2 jar, for example, /home/db2/sql1lib/java12/db2java.zip.
8. Click **Finish**.

To add a data source and define its resource properties:

1. If the server editor is not already open, double-click the server to open it. At the bottom of the editor, click the **Data Sources** tab.
2. Scroll down to the Server Settings section. In the JDBC provider list, select the JDBC provider you just created.
3. In the Server settings section, beside the **Data source defined in the JDBC provider selected above** list table, click **Add**. The Create a Data Source wizard opens. This is where you add your information for the dreamauc auction database that you created and populated when you earlier ran the SQL script.

4. In the JDBC Provider type list box, select your JDBC Provider:
  - For DB2: **DB2 Legacy CLI-based Type 2 JDBC Driver (XA)**
  - For Cloudscape: **Cloudscape JDBC Provider (XA)**
  - For iSeries: **DB2 UDB for iSeries (Toolbox XA)**
5. Click the **Version 5.0 data source** radio button and click **Next**. A version 5.0 data source is needed to run CMP 2.0 enterprise beans.
6. On the Modify Data Source page, type the following values:

Table 3. Data source values

Field	DB2	Cloudscape	iSeries
Name	Dream Auction	Dream Auction	Dream Auction
JNDI name	jdbc/dreamauc	jdbc/dreamauc	jdbc/dreamauc
Data source helper class name	com.ibm.websphere.rsadapter.DB2DataStoreHelper	com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper	com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper
Component-managed authentication alias	Alias that you defined earlier	N/A	Name of your iSeries userid
Container-managed authentication alias	N/A	N/A	Name of your iSeries userid

7. Select the **Use this data source in container managed persistence (CMP)** check box and click **Next**.
8. In the Resource Properties list table, select **databaseName**. (For iSeries, skip to step 10.)
9. In the **Value** field, type the following:
  - For DB2: Sample
  - For Cloudscape: the location of your Cloudscape dreamauc database. For example D:\temp\cloudscape\dreamauc.
10. For iSeries: In the Resource Properties list table, select **serverName**. In the **Value** field, type the name of your iSeries server.
11. Click **Finish**. In the steps you just completed, you associated a pooled database connection to a JNDI name that can be used by the application.

## Configuring the JMS settings

JMS is an API used to access enterprise message systems, much like JDBC is an API used to access databases. It allows the delivery of asynchronous messages; the JMS client can send a message without having to wait for a reply. You will use the point-to-point model, which consists of one message and one receiver. Now you will define the connection factory, queue, and listener port used by the application. To configure the JMS settings:

1. Click the JMS tab on the server editor.
2. Beside the **Queue Name** field, click **Add**. The Add Queue Name dialog box opens.
3. In the **Name** field, type BidProxyQ and click **OK**.
4. In the JMS Provider section, ensure that the **MQ Simulator for Java Developers** radio button is selected.

To add a queue connection factory:

1. In the JMS Connection Factories section, beside the WAS Queue Connection Factory list table, click **Add**. The Add WASQueueConnectionFactory dialog box opens.
2. In the **Name** field, type BidProxyQCF.
3. In the **JNDI Name** field, type jms/bidProxyQCF.
4. In the **Node** field, select **localhost**.
5. In the **Server name** field, select **server1** and click **OK**. The BidProxyQCF connection factory is defined.

To add a destination queue:

1. Beside the WAS Queue list table, click **Add**. The Add WASQueue dialog box opens.
2. In the **Name** field, type BidProxyQ.
3. In the **JNDI Name** field, type jms/bidProxyQ.
4. In the **Node** field, select **localhost** and click **OK**. The BidProxyQ name appears under the WASQueue entries. Your queue and topic connections and destinations appear in the JMS connection factories section of the JMS page.

To add a listener port:

1. Click the EJB tab of the server editor.
2. Beside the Listener Port list table, click **Add**. The Add Listener Port dialog box opens.
3. In the **Name** field, type P2PListenerPort.
4. In the **Connection factory JNDI Name** field, type jms/bidProxyQCF.
5. In the **Destination JNDI Name** field, type jms/bidProxyQ.
6. In the **Initial state** field, select **START**.
7. Click **OK**.
8. The P2PListenerPort is created. Save and close the editor.

### Overriding the client encoding (optional)


(Optional) If you are using a non-English system to operate WebSphere Studio, you must add a system property to the server configuration:

1. In the J2EE Hierarchy, expand **Servers**.
2. Double-click **TestServer**. The server configuration editor opens.
3. Click the Environment tab.
4. Beside the System Properties list table, click **Add**. The Add System Property dialog box opens.
5. In the **Name** field, type client.encoding.override.
6. In the **Value** field, type UTF-8 and click **OK**.
7. Save and close the editor.



## Test the business methods

Now you need to test your business methods using the universal test client. First you will launch the server on the DBConfig session bean, which will start the universal test client:

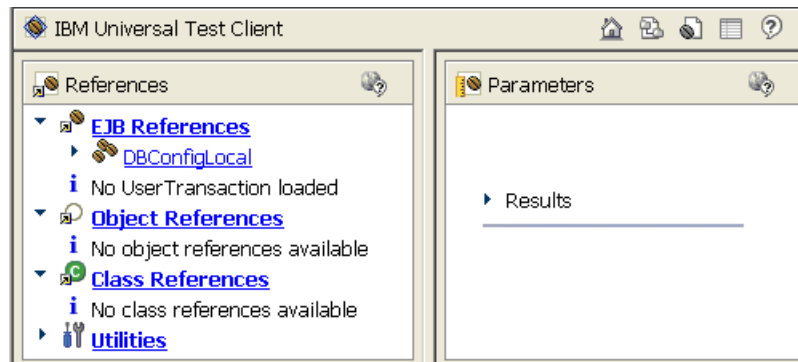
### Testing the DBConfig session bean

1.  From the **Window** menu, select **Preferences**. The Preferences page appears.



2.  In the left frame, select **Web Browser**, then in the **Location** field, type the following path for the Mozilla Web browser. For example, /usr/local/mozilla/mozilla
3.  Click **OK** to close the Preferences page.
4. In the J2EE Hierarchy view, expand **EJB Modules > AuctionEJBiz > Session Beans > DBConfig**.
5. Right-click **DBConfig** and select **Run on Server** to start the universal test client. The Server selection wizard opens.
6. Select **TestServer** and click **Finish**.
7. It may take a moment or two for the test client to appear, since all of the following tasks are being performed automatically:
  - Publishing the application to the test server
  - Configuring the test server
  - Starting the test server
  - Starting the test client
  - Starting the Web browser

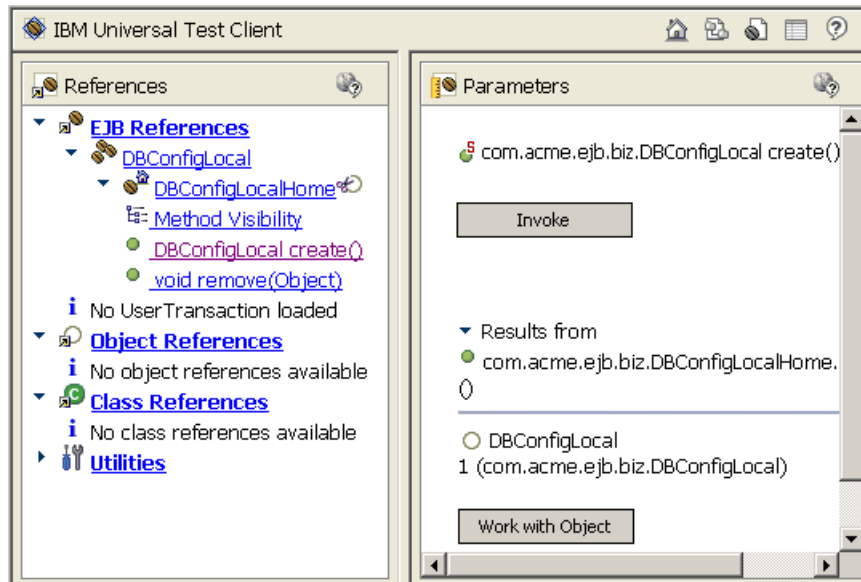
As the server starts, it displays messages in the Console view. Wait until the message Server server1 open for e-business is displayed. Since the server was launched on a specific enterprise bean, the Universal test client is opened on the bean in the browser window. The DB2ConfigLocal object appears in the Reference pane under the EJB References heading:



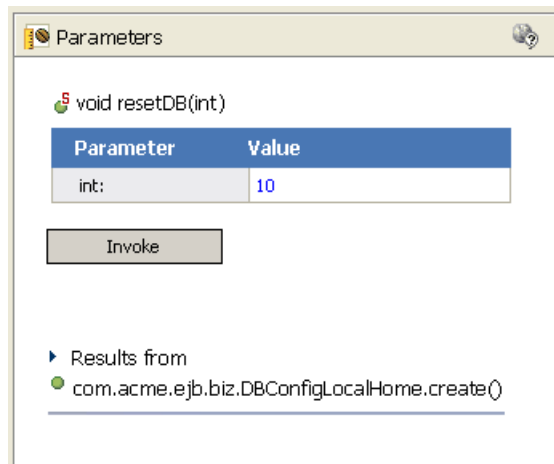
Now you will invoke methods and pass parameters to objects in order to populate the database tables.

1. In the References pane, fully expand **DBConfigLocal** to reveal the available methods on the DBConfig local home object.
2. Click **DBConfigLocal create()** link. This prepares the test client to invoke the create method.
3. In the Parameters pane, click **Invoke** to create and return a new DBConfig object. The DBConfig object is returned in the lower half of the Parameters pane:






4. Click **Work with Object**. This adds the DBConfigLocalHome instance to the References pane.
5. In the References pane, click the **void resetDB(int)** method link. The parameter displays a box where you can enter a value for the method parameter. This parameter is the expiration time of all bids in minutes.
6. In the Parameters pane, type 10 in the Value column:

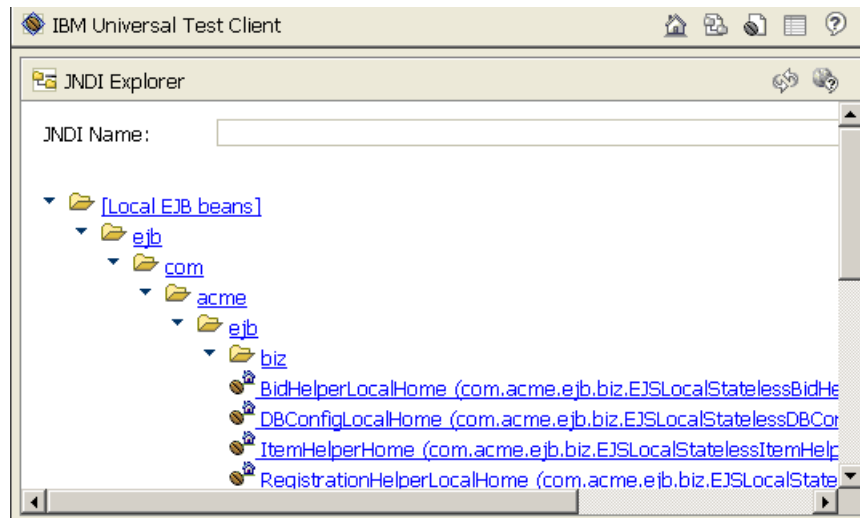


7. Click **Invoke**. The database tables have been populated with auction items that will expire in 10 minutes from now.

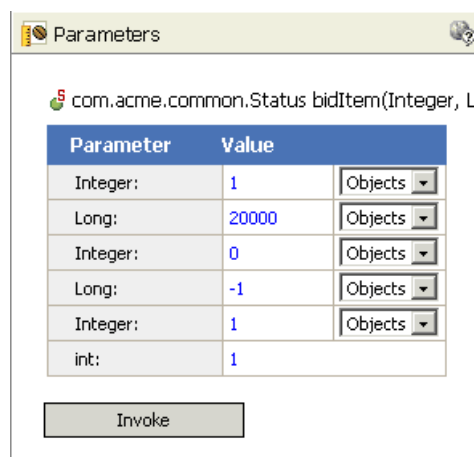
## Testing the ItemHelper session bean

Next, you will use the JNDI Explorer to test the ItemHelper session bean.

1. Above the Parameters pane, click the **JNDI Explorer** icon  to open the JNDI Explorer. You can immediately look up a JNDI resource by specifying its name in the JNDI field and selecting Lookup. Alternatively, you can expand a folder type in the list to locate your resource.
2. Expand the following folders: [Local EJB beans]/ejb/com/acme/ejb/biz.
3. Click the **BidHelperLocalHome** link:



4. The BidHelperLocal interface appears in the References pane of the test client. Click the **BidHelperLocalHome** link.
5. Click the **BidHelperLocal create()** method link. The object appears in the Parameters pane.
6. Click **Invoke** to invoke the method.
7. Click **Work with Object**.
8. In the References pane, click the **Status bidItem** link.
9. In the Parameters pane, type the following values in the Values column:
  - Integer: 1 (the item number)
  - Long: 20000 (your current bid in US cents)
  - Integer: 0 (your increment in US cents)
  - Long: -1 (maximum bid.-1 is used if only bidding once)
  - Integer: 1 (user ID in the Registration table)
  - int: 1 (currency)



10. Click **Invoke**. You should get a return code of 0.

### Testing the Onlineitem CMP bean


To test the Onlineitem bean:

1. Above the Parameters pane, click the **JNDI Explorer** icon to open the JNDI Explorer.

2. Click the **OnlineitemLocalHome** link. The Bean page opens.
3. In the References pane, expand **OnlineitemLocalHome** and click the **findByPrimaryKey(OnlineitemKey)** link.
4. In the Parameters pane, from the Constructors drop-down menu, select **OnlineitemKey(Integer)**.
5. Expand **OnlineItemKey** and **Constructor:OnlineitemKey**.
6. In the **Value** field for the Integer row, type 1 and click **Invoke**. (In these steps, you are creating a new key and then passing the key into the **findByPrimaryKey** method.)


Parameter	Value
OnlineitemKey:	<a href="#">Collapse</a>
Constructor:OnlineitemKey:	
Integer:	1

Invoke

7. Click **Work with Object**. This places the item that you found in the previous step in the References pane.
8. In the References pane, click the **String getLastbidder()** method.
9. In the Parameters pane, click **Invoke**.
10. The name of the last bidder appears (John Doe).
11. In the References pane, click the **Long getLastbid()** method.
12. In the Parameters pane, click **Invoke**.
13. The value of the last bid appears (20000 cents).
14. If you want, you can continue to explore the features of the test client by invoking more methods.
15. When you are finished exploring, in the Servers perspective, click the **Servers** tab to switch to the Servers view.
16. In the Server column, select the **TestServer** server and then click the **Stop** icon .
17. Close the browser session where the Universal Test Client is running.

## Create the Web Service

Now that you have finished creating the enterprise beans for the Auction application, you will create a simulated credit card company called **DreamCreditCard**. You will create a Web service to approve the transactions sent from the Auction application. To create the enterprise application project that will hold the Web service:

1. Click the **Open The New Wizard** icon  at the left end of the toolbar.
2. In the left frame of the New wizard, select **J2EE**.
3. In the right frame, select **Enterprise Application Project** and click **Next**.
4. Select the **Create J2EE 1.3 Enterprise Application project** radio button and click **Next**.
5. In the **Project name** field, type **DreamCreditCard**.

6. Click **Next**.
7. Click the **New Module** button. The New Module Project wizard opens.
8. Clear the **Application Client Project** and the **Connector Project** check boxes and click **Finish**.
9. Click **Finish** again.

The business logic of the Web service is provided for you in an EJB JAR file. To import the EJB project into the DreamCreditCard enterprise application project:

1. From the **File** menu, select **Import** to open the Import wizard, then select **EJB JAR file** and click **Next**.
2. Beside the **EJB JAR file** field, click **Browse** and then navigate to the following file and select it (where *WS\_installdir* is the directory where you have installed WebSphere Studio):

 Windows

*WS\_installdir*\samples\scenario\_parts\auction\v51\DreamCreditCardEJB.jar

 Linux

*WS\_installdir* /samples/scenario\_parts/auction/v51/DreamCreditCardEJB.jar

Click **Open**.

3. In the **EJB Project** field, select **DreamCreditCardEJB**.
4. (Optional) If you have enabled server targeting, in the **Target Server** field, select **WebSphere Application Server v5.1**.
5. Click **Finish**. The files contained in DreamCreditCardEJB.jar are imported into the DreamCreditCard EJB project. If prompted to overwrite the META-INF/MANIFEST.MF file, click **Yes**.
6. To generate deployment code, right-click **DreamCreditCardEJB** and select **Generate > Deployment and RMIC code**.
7. Click **Select all** to mark all of the beans for code generation.
8. Click **Finish** to generate the deployment and RMIC code.

## Create the EJB Web service

We will create a new Web service based on the existing DreamCreditCardEJB. The Web service allows the bids to be processed through a credit card company. To create the Web service:

1. In the Project Navigator view, right-click the DreamCreditCardWeb project and select **New > Other > Web Services > Web Service**. The Web Services wizard opens.
2. In the **Web service type** field, select **EJB Web service**.
3. Select the **Generate a proxy** check box. The generated proxy invokes methods from the Web service for testing purposes.
4. Select the **Test the generated proxy** check box and click **Next**.
5. The Service Deployment Configuration page allows you to choose which run-time protocol and deployment server to use. In the Server-Side Deployment Selection section, click **Edit**. The Service Deployment Configuration dialog box opens.
6. Select the **Choose server first** radio button.
7. In the Server selection list, expand **Existing servers** and select **TestServer**.
8. In the Web service run time list, select **IBM WebSphere v5**.
9. In the **EJB Project** field, select **DreamCreditCardEJB**.

10. In the **Router Project** field, select **DreamCreditCardWeb** and click **OK**.
11. The Client-side Environment Selection configuration settings should default to the same run time and server as the Server-Side Deployment configuration settings. In the Client-side Environment Selection section of this wizard page, ensure that the Web service run time is IBM WebSphere v5 and that the server is TestServer.
12. In the **Client Web Project** field, type DreamCreditCardWebClient and click **Next**.
13. The Web Service EJB Selection page allows you to choose which enterprise bean you want to configure as a Web service. Click the **Browse EJB beans** button. The Browse EJB Beans dialog opens.
14. In the **EAR projects** field, select **DreamCreditCard**.
15. In the Stateless EJB beans column, select **TransferFunds** and click **OK**.
16. Ensure the **SOAP over HTTP** radio button is selected.
17. Accept the default values on the following pages of the wizard and click **Next** until the last page of the wizard where you can click **Finish**.

The Web service wizard generated a sample application that executes methods from the Web service. This application demonstrates the structure of a client proxy. To test the Web service:

1. In the Methods pane, click the creditCardPurchase method. This brings up a form for entering the method's parameters in the top-right pane.
2. In the Inputs pane, fill in the fields with the following values:
  - **merchantAcct:** 1234
  - **merchantPassword:** 1234
  - **cardNumber:** 1111222233334444
  - **expDate:** 12/2005
  - **purchaseDate:** 12/25/03
  - **amount:** 20
  - **desc:** Tulips
3. Click **Invoke**. The results of invoking the method appear in the Result pane, with the transaction date, amount, description and credit card number.
4. In the Servers view, select **TestServer** and click the **Stop** button.

To set up the Web service in the AuctionEJBBiz project:

1. In the Project Navigator view, expand **DreamCreditCardWebClient > Java Resources**.
2. Select both **com.dcc.account** and **com.dcc.account.ejb**. Right-click these folders and select **Copy**.
3. Expand **AuctionEJBBiz > ejbModule**. Right-click and select **Paste**. If you have enabled server targeting, skip steps 4–10.
4. Right-click **AuctionEJBBiz** and select **Properties**.
5. In the left frame, select **Java Build Path**.
6. In the right frame, select the Libraries tab and click the **Add Variable** button. The New Variable Classpath Entry dialog box opens.
7. Select the **WAS\_50\_PLUGINDIR** variable and click **Extend**. The Variable Extension dialog box opens.
8. Expand lib and select the following JAR files:
  - qname.jar

- webservices.jar
  - wsdl4j.jar
9. Click **OK** to close the dialog box.
  10. Click **OK** again to close the properties page.
  11. In the com.acme.ejb.biz folder, open ItemHelperBean.java.
  12. Uncomment the following two import statements:
 

```
//import com.dcc.account.TransactionConfirmation;
//import com.dcc.account.ejb.TransferFundsServiceLocator;
```

and the lines in the method

```
public Status processExpiredItems() {
```

**Tip:** Highlight the commented lines and from the main menu select **Source > Uncomment** to uncomment the lines.

13. Save and close the editor.


To simplify the setup of the auction Web site, we will run both the Auction and DreamCreditCard enterprise applications on the same server. To simulate a real production environment, you can run the DreamCreditCard enterprise application on a separate server. You will need to edit the TransferFunds\_address variable in the TransferFundsServiceLocator.java file of the AuctionEJBiz project so that it corresponds to the server that runs the DreamCreditCard application. Alternatively, you can regenerate the Web service using the hostname of DreamCreditCard in the JNDI provider URL.

## Set up the team environment

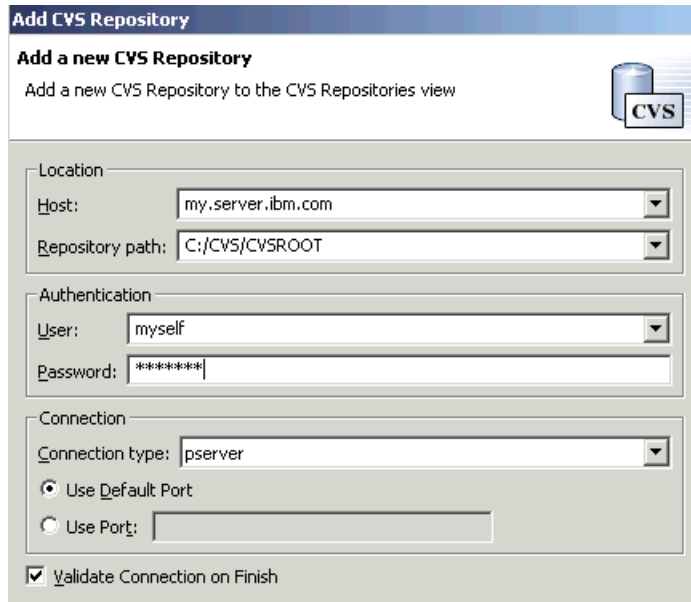
In this optional section, you configure your workspace with the location of the team server. (This section assumes that you either have access to a CVS server or you have already installed a CVS server for yourself using the instructions in the topic “Preparing for the Online Auction sample application”).

1. Open the CVS Repository Exploring perspective (**Window > Open Perspective > Other > CVS Repository Exploring**).
2. Position your mouse pointer in the CVS Repositories view, then right-click and select **New > Repository Location**. The Add CVS Repository wizard appears. (To determine the actual values that you need to specify in the wizard, you will probably need to talk to your CVS administrator.)
3. In the **Host** field, type the host name or IP address of the team server.
4. In the **Repository path** field, type the fully qualified directory name of the CVS repository on the server. For example:

 Windows  
D:\CVS\CVSROOT

 Linux  
/usr/local/cvsroot

5. In the **User** field, type in your team server user name.
6. In the **Password** field, type in your CVS password
7. In the **Connection type** field, ensure that the correct connection type is selected:



**Add CVS Repository**

Add a new CVS Repository to the CVS Repositories view

Location

Host: my.server.ibm.com

Repository path: C:/CVS/CVSROOT

Authentication

User: myself

Password: \*\*\*\*\*

Connection

Connection type: pserver

☒ Use Default Port

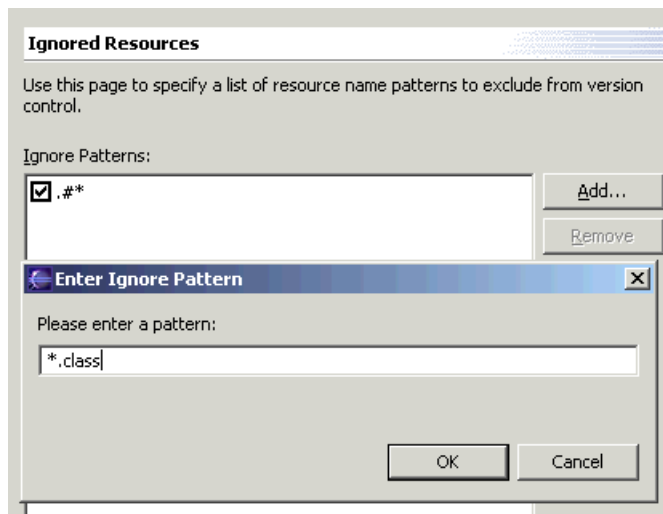
☐ Use Port:

☒ Validate Connection on Finish

8. Click **Finish**.

To prevent the class files from being stored in CVS, and save space in the repository:

1. From the main menu, click **Window > Preferences > Team > Ignored Resources**.
2. Click the **Add** button beside the Ignore Patterns pane. The Enter Ignore Pattern dialog box opens.
3. Type \*.class and click **OK**.



**Ignored Resources**

Use this page to specify a list of resource name patterns to exclude from version control.

Ignore Patterns:

☒ .#\* Add... Remove

**Enter Ignore Pattern**

Please enter a pattern:

\*.class

OK Cancel

4. Click **OK** on the Preferences page to save your changes.


You can specify label decorations to appear for CVS resources, to show CVS specific information on resources under CVS control. To display CVS label decorations:

1. From the main menu, click **Window > Preferences > Workbench > Label Decorations**.
2. In the Available label decorations list box, click the **CVS** check box.

3. Click **OK** on the Preferences page to save your changes.

## Version the application

In this optional section, you learn how to version your application. (This section assumes that you either have access to a CVS server or you have already installed a CVS server for yourself using the instructions in the topic “Preparing for the Online Auction sample application”).

1. Switch to the J2EE perspective.
2. In the Project Navigator view, select the **Auction** project.
3. Right-click and select **Team > Share Project**. The Share Project wizard opens.
4. (Optional: This step is only required if you have the ClearCase® plugin installed). Select **CVS** and click **Next**.
5. Select the existing repository and click **Next**.
6. Click the **Use project name as module name radio button** and click **Next**.
7. Click **Finish** to import the project into the CVS repository. The Synchronize view opens.
8. In the Structure Compare pane, right-click **Auction** and then select **Commit**.
9. You are prompted to add the resources to version control. Click **Yes**. (If you want, you can enter a comment in the next dialog box when prompted.)
10. Repeat steps 1–9 for each of the AuctionEJB, AuctionEJBiz, AuctionWeb, DreamCreditCard, DreamCreditCardEJB, DreamCreditCardWeb, and DreamCreditCardWebClient projects. The server is updated with a file structure that matches the structure you have in the local file system and the files on the server are automatically compared to the files in the local file system.
11. Switch to the CVS Repository Exploring perspective.
12. In the CVS Repositories view, click the **Refresh View** icon . Expand the connection and expand HEAD.
13. Select the **Auction**, **AuctionEJB**, **AuctionEJBiz**, **AuctionWeb**, **DreamCreditCard**, **DreamCreditCardEJB**, **DreamCreditCardWeb**, and **DreamCreditCardWebClient** projects together, then right-click and select **Tag as Version**. This causes the server to associate the same tag for all the files in these three projects on the server. The Tag Resources dialog box appears.
14. In the **Please enter a version tag** field, type V1 and click **OK**.
15. In the CVS Repositories view, expand **Versions** and then expand **Auction** to see the new version.

## Add shared projects to another workspace

In this optional section, you learn how to load the application into a different WebSphere Studio workspace that resides on a remote machine, such as a Web designer’s machine. If you don’t have access to a remote machine, instructions are provided for simulating the experience by opening up a second workspace on your own local machine. In this section and the following section, the second workspace is referred to as the “Web designer’s workspace.” (This section assumes that you either have access to a CVS server or you have already installed a CVS server for yourself using the instructions in the topic “Preparing for the Online Auction sample application”).

1. If you are using a remote machine, follow the instructions given previously for installing DB2 on the remote machine and running the DB2 script that is used to create the required database tables.



2. If you do not have access to a remote machine and you want to open up a second workspace on your local machine instead, complete the following three steps:

- a. Close your current workspace by exiting WebSphere Studio.
- b. Start WebSphere Studio. You are prompted to specify a directory for your workspace. Type the location of a new workspace. For example:

Windows

`workspace_path\workspace2`

Linux

`/home/user_id/workspace_path/workspace2`




WebSphere Studio opens with the new workspace. None of the resources that you created in your other workspace are currently visible.

3. Open the CVS Repository Exploring perspective.
4. Position your mouse pointer in the CVS Repositories view, then right-click and select **New > Repository Location**. The Add CVS Repository wizard appears. (To determine the actual values that you need to specify in the wizard, you will probably need to talk to your CVS administrator.)
5. In the **Host** field, type the host name or IP address of the team server.
6. In the **Repository path** field, type the fully qualified directory name of the CVS repository on the server. For example:  

Windows `D:\CVS\CVSROOT`  
Linux `/usr/local/cvsroot`
7. In the **User** field, type in your team server user name.
8. In the **Password** field, type in your CVS password.
9. In the **Connection type** field, ensure that the correct connection type is selected and click **Finish**.
10. Add CVS label decorations as previously described in the Set up the team environment section.
11. The default settings in WebSphere Studio are set to prune empty directories in CVS. To disable this setting, from the main menu, click **Window > Preferences > Team > CVS**.
12. Clear the **Prune empty directories** check box and click **OK** to exit the Preferences page.
13. In the CVS Repositories view, expand the new repository node and expand **HEAD**, then select **Auction**, **AuctionEJB**, **AuctionEJBBiz**, **AuctionWeb**, **DreamCreditCard**, **DreamCreditCardEJB**, **DreamCreditCardWeb**, and **DreamCreditCardWebClient** together. Right-click and select **Check out As Project**. This adds the projects to the Web designer's workspace.

To create and configure a server:



1. Right-click the Servers folder in the J2EE Hierarchy view. Select **New > Server and Server Configuration**. The Create a New Server and Server Configuration wizard opens.
2. In the **Server Name** field, type TestServer.
3. In the **Folder** field, type Servers.
4. In the **Server type** list box, expand **WebSphere version 5.1** and select **Test Environment**.
5. Click **Finish** to create the server.
6. Linux In the J2EE Hierarchy, expand **Servers**.

7.  Double-click **TestServer**. The server editor opens.
8.  For DB2: In the server editor, click the **Environment** tab, then click **Add External JARs** in the **WebSphere specific class path (ws.ext.dirs)** section to add **DB2\_installdir/java12/db2java.zip** to the WebSphere path, where **DB2\_installdir** is the directory where DB2 is installed.  
For iSeries: In the server editor, click the **Environment** tab, then click **Add External JARs** in the **WebSphere specific class path (ws.ext.dirs)** section to add the **jt400.jar** file to the path.
9.  Click **File > Save (name of file)** to save your changes.

Now you need to add a data source so that the server can access the data in the dreamauc database. You will need to complete the following tasks:

- Defining a JAAS authentication entry
- Defining a JDBC provider
- Defining a data source
- Defining resource properties for the data source, such as userids and passwords


To define a JAAS authentication entry (skip this section if you are using Cloudscape):

1. In the J2EE Hierarchy view, expand **Servers**.
2. Double-click **TestServer**. The server editor opens.
3. At the bottom of the editor, click the **Security** tab.
4. In the JAAS Authentication Entries section, click **Add**. The JAAS Authentication Entry dialog opens.
5. In the Alias field, type your database user ID.  For Linux this is **db2inst1** for a DB2 database.  
For iSeries: Type the name of the iSeries server.
6. In the User ID field, type your database user ID again.  For Linux this is **db2inst1** for a DB2 database.  
For iSeries: Type your iSeries user ID.
7. In the Password field, type your database password then click **OK**.  
For iSeries: Type your iSeries password.

To create a JDBC provider:

1. At the bottom of the editor, click the **Data source** tab.
2. Scroll down to the Server Settings section. Beside the JDBC provider list table, click **Add**. The Create a JDBC Provider wizard opens.
3. In the **Database type** list box, select **IBM DB2** or **Cloudscape**.
4. In the **JDBC provider type** list box, select one of the following provider types and click **Next**:
  - For Cloudscape: **Cloudscape JDBC Provider (XA)** You can safely ignore the description stating that this provider is deprecated.
  - For DB2: **DB2 Legacy CLI-based Type 2 JDBC Driver (XA)**
  - For iSeries: **DB2 UDB for iSeries (Toolbox XA)**

The XA provider supports the two-phase commit protocol, which is when multiple changes across multiple resources such as beans can be treated as a single transaction. This is required by the BidHelper session bean as it will update the database via the CMP beans and also send a JMS message in a single transaction.

5. In the **Name** field, type a name for your JDBC provider, for example, DB2 JDBC Type 2 Provider (XA) or iSeries Toolbox XA.
6. For iSeries: Click the **Remove** button beside the **Class path** list box to delete the default entry. Beside the Class path list box, click **Add External JARs** and browse to the **jt400.jar**.
7.  For DB2: For DB2: Click the **Remove** button beside the **Class path** list box to delete the default entry. Beside the Class path list box, click **Add External JARs** and browse to the DB2 jar, for example, /home/db2/sqllib/java12/db2java.zip.
8. Click **Finish**.

To add a data source and define its resource properties:

1. If the server editor is not already open, double-click the server to open it. At the bottom of the editor, click the **Data Sources** tab.
2. Scroll down to the Server Settings section. In the JDBC provider list, select the JDBC provider you just created.
3. In the Server settings section, beside the **Data source defined in the JDBC provider selected above** list table, click **Add**. The Create a Data Source wizard opens. This is where you add your information for the dreamauc auction database that you created and populated when you earlier ran the SQL script.
4. In the JDBC Provider type list box, select your JDBC Provider:
  - For DB2: **DB2 Legacy CLI-based Type 2 JDBC Driver (XA)**
  - For Cloudscape: **Cloudscape JDBC Provider (XA)**
  - For iSeries: **DB2 UDB for iSeries (Toolbox XA)**
5. Click the **Version 5.0 data source** radio button and click **Next**. A version 5.0 data source is needed to run CMP 2.0 enterprise beans.
6. On the Modify Data Source page, type the following values:

Table 4. Data source values

Field	DB2	Cloudscape	iSeries
<b>Name</b>	Dream Auction	Dream Auction	Dream Auction
<b>JNDI name</b>	jdbc/dreamauc	jdbc/dreamauc	jdbc/dreamauc
<b>Data source helper class name</b>	com.ibm.websphere. rsadapter. DB2DataStoreHelper	com.ibm.websphere. rsadapter. Cloudscape DataStoreHelper	com.ibm.websphere. rsadapter. DB2AS400Data StoreHelper
<b>Component-managed authentication alias</b>	Alias that you defined earlier	N/A	Name of your iSeries userid
<b>Container-managed authentication alias</b>	N/A	N/A	Name of your iSeries userid

7. Select the **Use this data source in container managed persistence (CMP)** check box and click **Next**.
8. In the Resource Properties list table, select **databaseName**. (For iSeries, skip to step 10.)
9. In the **Value** field, type the following:
  - For DB2: Sample
  - For Cloudscape: the location of your Cloudscape dreamauc database. For example D:\temp\cloudscape\dreamauc.

10. For iSeries: In the Resource Properties list table, select **serverName**. In the **Value** field, type the name of your iSeries server.
11. Click **Finish**. In the steps you just completed, you associated a pooled database connection to a JNDI name that can be used by the application.

### Configuring the JMS settings

To configure the JMS settings:

1. Click the JMS tab on the server editor.
2. Beside the **Queue Name** field, click **Add**. The Add Queue Name dialog box opens.
3. In the **Name** field, type BidProxyQ.
4. In the JMS Provider section, ensure that the **MQ Simulator for Java Developers** radio button is selected.

It is a recommended practice to configure the server at the individual server level which provides application isolation, rather than the cell or node level. To add a queue connection factory:

1. In the Server Settings section, beside the WAS Queue Connection Factory list table, click **Add**. The Add WASQueueConnectionFactory dialog box opens.
2. In the **Name** field, type BidProxyQCF.
3. In the **JNDI Name** field, type jms/bidProxyQCF.
4. In the **Node** field, select **localhost**.
5. In the **Server name** field, select **server1** and click **OK**.

To add a queue:

1. Beside the WAS Queue list table, click **Add**. The Add WASQueue dialog box opens.
2. In the **Name** field, type BidProxyQ.
3. In the **JNDI Name** field, type jms/bidProxyQ.
4. In the **Node** field, select **localhost** and click **OK**. Your queue and topic connections and destinations appear in the JMS connection factories section of the JMS page.

To add a listener port:

1. Click the EJB tab of the server editor.
2. Beside the Listener Port list table, click **Add**. The Add Listener Port dialog box opens.
3. In the **Name** field, type P2PListenerPort.
4. In the **Connection factory JNDI Name** field, type jms/bidProxyQCF.
5. In the **Destination JNDI Name** field, type jms/bidProxyQ.
6. In the **Initial state** field, select **START**.
7. Click **OK**.
8. Save and close the editor.

### Overriding the client encoding (optional)

(Optional) If you are using a non-English system to operate WebSphere Studio, you must add a system property to the server configuration:

1. In the J2EE Hierarchy, expand **Servers**.
2. Double-click **TestServer**. The server configuration editor opens.
3. Click the Environment tab.

4. Beside the System Properties list table, click **Add**. The Add System Property dialog box opens.
5. In the **Name** field, type `client.encoding.override`.
6. In the **Value** field, type UTF-8 and click **OK**.
7. Click **File > Save TestServer** to save the configuration, then close the editor.

## Add a professional Web front end

In this section, you add a Web front end to your application that will include the following functions:

- Searching for online items (based on a specified value)
- Obtaining details of a particular online item
- Registering an item for sale in the auction
- Bidding for an item

To save you time, the Web front end is already provided for you in a WAR file, which contains the artwork, JSP files, and servlets.

To add the Web front end:

1. If you completed the steps in the previous section “Add shared projects to another workspace”, then remain in your current Web designer’s workspace to complete the steps in this section. However, if you did not complete the steps in the “Add shared projects to another workspace” section, you should continue to use the workspace that you have been using up to this point in the tutorial.
2. From the **File** menu, select **Import** to open the Import wizard, then select **WAR file** and click **Next**. A Web archive (WAR) file is a packaged Web application that is deployed on a Web server. A Web application is a group of HTML pages, JSP pages, servlets, and other resources, along with source files that can be managed as a single unit.
3. Beside the **WAR File** field, click **Browse** and then navigate to the following WAR file and select it (where `WS_installdir` is the directory where you have installed WebSphere Studio):
 

Windows

`WS_installdir/samples/scenario_parts/auction/v51/AuctionWeb.war`  

Linux

`WS_installdir/samples/scenario_parts/auction/v51/AuctionWeb.war`
4. In the **Project** field, select **AuctionWeb**.
5. Select the **Overwrite existing resources without warning** check box.
6. Click **Finish**. The WAR file is imported into the AuctionWeb project. Any errors or warnings that appear in the Task list will be resolved as you complete the instructions
7. To set the module dependency for the AuctionWeb project to the AuctionEJBBiz project, right-click the AuctionWeb project and select **Properties**. The Properties page opens.
8. On the left frame of the Properties page, click **Java JAR Dependencies**.
9. In the Available dependent JARs list table, click the **AuctionEJBBiz.jar** check box and click **OK**. This enables the Web project to access the classes in the EJB project at run time.
10. If you completed the optional “Set up the team environment” and “Version the application” sections, then complete the following steps:
  - a. In the **Project Navigator** view, right-click the **AuctionWeb** project, then select **Team > Synchronize with Repository**. The Synchronize view opens.

- b. In the Structure Compare pane, right-click **AuctionWeb** and then select **Commit**.
  - c. You are prompted to add the resources to version control. Click **Yes**. (If you want, you can enter a comment in the next dialog box when prompted.)
11. Now you have a fully functioning application and you will test the application in the next section.

### Testing the application on a server

In this section, you take a short guided tour through the Online Auction Web site. Note that only some of the options and features in the application are actually operational. For example, some of the categories in the Category List are simply cosmetic enhancements to the Web pages and perform no actual function. To start the application:

1. In the J2EE Hierarchy view, generate deployment code for the AuctionEJB project.
2. Generate deployment code for the AuctionEJB Biz project.
3. Generate deployment code for the DreamCreditCardEJB project.
4. In the Project Navigator view, right-click the **AuctionWeb** project and select **Run on Server**.
5. Click the **Use an existing server** radio button, select **TestServer** and click **Finish**.
6. The Online Auction home page opens in the Web browser (although this may take a moment or two).

To take a tour of the Auction web site:

1. Before you can sell or bid on any items, you need to prepare the data. Click the **setup** link in the header bar.
2. Enter the time (in minutes) for how long you want the items to be in the auction before they expire. The default time is 4 minutes.
3. Click the **Initialize Database** button to set up the auction items. The items are displayed on the Floral page. Now you can bid on any items which are not expired.
4. In the header bar, click **sell**. The Sell an Item page is displayed. In the Sell an Item page, populate the fields with some values. For example:
  - **Seller:** Select an email address from the drop-down list
  - **Title:** Ancient painted flower vase
  - **Minimum bid:** 150
  - **Value:** 220
  - **Description:** This 100-year old vase must be seen in person!
  - **End bidding time** 5
5. If you would like to include images, select the check box at the bottom and click **Browse** to select an image
6. Click **Submit**. The item you want to sell is registered in the auction and the Arts and Crafts Floral page opens and displays the item that you registered for sale.
7. In the right frame of the Arts and Crafts Floral page, type 250 in the search by price field.
8. Click the **Go** button. All items with a value of \$250 or less (if any) are displayed.



9. In this next step, you move from the role of seller to the role of bidder and bid on the item you just registered for sale. To bid on an item, click the title of the item or the **Bid** button beside it. The bid page opens.
10. To simplify the setup of the auction, there is a pre-populated list of users that identifies potential bidders. You do not have to log into the site to place a bid. Select an email address from the pre-populated list of users.
11. The bid increment is set to 5% of the value of the item. The next available bid is the next lowest bid that you can place.. You can also place an auto bid. In the Maximum Bid field, enter the maximum amount you want to pay for an item and click the **Auto Bid** button. Once you have placed an auto bid, the auction site will place a new bid on your behalf when someone has bid over your current bid amount, until a bid reaches the maximum value. This is implemented by the BidProxy message-driven bean you created earlier.
12. Your bid is submitted and the Web page is refreshed to show your previous bid value and the name you used to submit the bid. You can then specify another bid under a different email address to simulate another user. You can monitor the bidding activities by looking at the output in the Console view.
13. After one or more items have expired, the winning transaction will be processed through the Web service provided by the winner's credit card company. To initiate this process, click the **setup** link in the header bar. The setup page opens. Click the **Process Expired Items** button. The transactions are processed using the DreamCreditCard Web service that you created earlier. The transaction details are displayed in the Console view.
14. At the bottom of the workspace, click the Servers tab.
15. In the Server column, select the **TestServer** server and then click the **Stop** icon.

**Note:** This auction site supports making bids and viewing items in different currencies. The currency that you see depends on the Locale setting of your computer. If your locale is set to en\_CA (i.e. Canada) you will see monetary amounts in Canadian dollars (e.g. C\$15.50). If your locale is set to en\_IE (Ireland) you will see monetary amounts in Euros (e.g. €12.50). All other locales will show US dollars (e.g. \$11.00)

Congratulations! You have successfully run the Online Auction application. If you used a second workspace (the Web designer's workspace, at this point in the scenario you should now close the second workspace and open your original developer's workspace by exiting and restarting WebSphere Studio with your original workspace. Complete the following two steps:

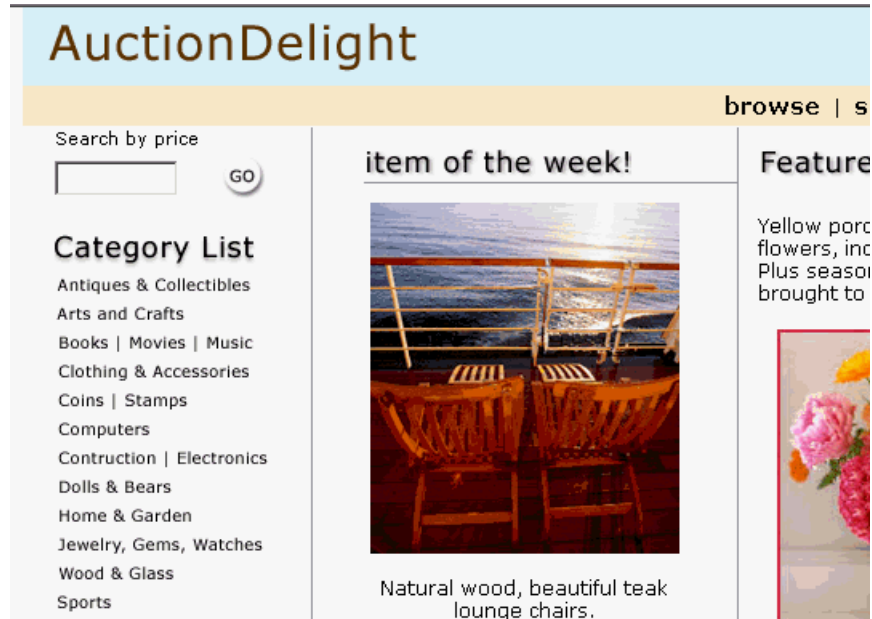
1. Switch to the J2EE perspective. In the Project Navigator view, right-click **AuctionWeb** and select **Team > Synchronize with Repository**.
2. In the Structure Compare pane, right-click AuctionWeb and select **Update from Repository**. This moves all of the work that was done in the Web designer's workspace into the original developer's workspace.

## Debug the application



To debug the application in the test environment:

1. To set debugging preferences, from the main menu click **Window > Preferences > WAS Debug**.
2. Clear the **Use step-by-step debug mode** check box and click **OK**.
3. Right-click the **AuctionWeb** project and select **Debug on Server**. The Server selection wizard opens.

4. Click the **Use an existing server** radio button.
5. Select **TestServer** and click **Finish**.
6. The entire application is automatically deployed and the Online Auction home page opens up in the browser window (which may take a moment or two). The URL is `http://localhost:9080/AuctionWeb`:



Now that you have the application running in the test environment, you can set breakpoints and debug the application while it is running. To learn about how search results are displayed in the application and debug a JSP file:

1. In the Project Navigator view, expand **AuctionWeb** and **WebContent**, then double-click **FloralResults.jsp** to open the JSP file in the source editor.
2. Click the **Source** tab and then locate the following line of code (around line 55):  
**Tip:** Use **Ctrl + L** to go to a specific line.  
`com.acme.common.SearchResultBean r = results[i];`
3. Right-click the left margin beside the line of code, and select **Add breakpoint** from the pop-up menu.
4. Switch back to **Web Browser**.
5. In the **search by price** field, type 150 and then click the **Go** button.
6. The Debug perspective opens and **FloralResults.jsp** is suspended at the breakpoint.
7. In the **Variables** view, browse some of the variables in the JSP file.
8. In the **Debug** view, click the **Step Over** icon  to step through some of the code.
9. Right-click the breakpoint and select **Remove Breakpoint**.
10. Click the **Resume** icon  to finish running the program. (Note that if you need to make changes to your JSP files or to other classes in the Web project, the test environment will detect the changes and automatically reload the files or classes.)



In this release, the WebSphere V5 test environment supports hot method replace when the server is running in debug mode. To learn how the business methods work, such as the `bidItem` method:

1. From the main menu, click **Window > Show View > Other > J2EE > Project Navigator**. The Project Navigator view opens.
2. In the Project Navigator view, expand the **AuctionEJB**, **ejbModule**, and **com.acme.ejb.biz** folders, then double-click **BidHelperBean.java** to open the source editor.
3. Set a breakpoint beside the following line in the `bidItem` method (and ensure that you put it in the `bidItem` method, since the line also appears in other methods):  

```
Status status = new Status();
```
4. Switch back to **Web Browser** and then click the round **Bid** button beside one of the flower items for sale.
5. In the **Your Name** field, select an email address from the drop-down menu.
6. In the **Maximum Bid** field, type 250 and click the **Auto Bid** button.
7. In the Debug view, expand the suspended thread that has the + symbol beside it, then click the **BidHelperBean.bidItem** line.
8. Examine some variables in the Variables view.
9. In the Debug view, single-step through some code by clicking the **Step Over** icon.
10. You may insert a new line of code such as  

```
System.out.println("Test Hot Method Replace");
```

  
into the method.
11. When you save the changes, the execution will be reverted back to the first line of the current method. You can continue to step over the lines to see the execution of the newly added line.  
**Note:** The default transaction timeout is 120 seconds. The transaction will be rolled back if the execution is suspended longer than the timeout value. The timeout value can be increased in the **Transaction Timeout** field of the EJB page of the server editor.
12. Remove the breakpoint that you set in the source code.
13. In the Debug view, click the **Resume** icon.
14. Close the open files and the Web browser.
15. Switch to the J2EE perspective and click the **Servers** tab to open the Servers view.
16. In the Server column, select the **TestServer** server and then click the **Stop** icon.
17. If you plan to run the ready-made sample application (as described in Part 1) after building the sample application for yourself from scratch, then you need to remove the Auction project from the server by completing the following steps:
  - a. In the Servers view, right-click **TestServer** and select **Add and remove projects**
  - b. In the Configured projects pane, select **Auction** and click **Remove**.
  - c. Click **Finish**.

(Note that if you make changes to your enterprise bean code, you will need to right-click the EAR project and select **Restart Project**.)

Congratulations! You have built the Online Auction sample application and run it in the WebSphere test environment. You learned how to create various types of enterprise beans that hold the business logic of the application, create and configure servers that connect to a database, test the business methods in the test client, wrap an enterprise bean in a Web service, work in a team environment using a CVS repository, and test and debug the application in the WebSphere test environment.

---

## Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan*

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

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*Lab Director  
IBM Canada Ltd. Laboratory  
8200 Warden Avenue  
Markham, Ontario, Canada L6G 1C7*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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

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

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

#### COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (C) Copyright IBM Corp. 2000, 2003. All rights reserved.

---

## Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

**Warning:** Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

---

## Trademarks and service marks

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

- AIX
- CICS
- Cloudscape
- DB2
- DB2 Extenders
- DB2 Universal Database
- e-business
- IBM
- iSeries
- OS/390
- S/390
- VisualAge
- WebSphere
- z/OS

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

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

UNIX is a registered trademark of The Open Group.

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation, in the United States, other countries or both.

Other company, product, and service names, which may be denoted by a double asterisk(\*\*), may be trademarks or service marks of others.