Inside WebObjects

Developing EJB Applications

January 2002

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada

Contents

Figures, Listings, and Tables

About This Book

Enterprise JavaBeans (EJB) is a specification that provides an infrastructure through which solution providers can develop components that you can purchase and use in your WebObjects applications with minimal effort. In addition, the components can be configured to work with a variety of databases (as long as the database supports JDBC). The key ingredient in these components is enterprise beans. Enterprise beans are business objects that contain logic used to perform specific tasks. They are similar to enterprise objects in WebObjects, but can be used in application servers by multiple vendors.

Enterprise JavaBeans is part of Sun's Java 2 Platform, Enterprise Edition (J2EE) strategy. J2EE provides an abstraction from the implementation details of databases, directory services, communication protocols, and so on. EJB aims at providing you an abstraction layer between your application's business logic and the implementation-specific details of the data entities it uses. An enterprise-bean developer doesn't have to worry about which database is used when the bean is deployed, freeing her to concentrate on the business problem. WebObjects implements version 1.1 of the EJB specification.

Enterprise JavaBeans support in WebObjects lets you integrate third-party, enterprise-bean–based solutions in your WebObjects applications. This means you can purchase components that solve a particular problem, so that you can focus on issues specific to your business. In addition, you can develop your own enterprise beans using WebObjects tools. You must keep in mind, however, that WebObjects's Enterprise Object technology does not complement, nor can be efficiently combined with Enterprise JavaBeans. When you write enterprise beans, you use a persistence-management system that is completely separate from Enterprise Objects. You should not have enterprise beans that use the same database tables that enterprise objects are mapped to.

About This Book

You should read this book if you want to learn how to incorporate an EJB-based solution in a WebObjects application or you want to develop your own enterprise beans using WebObjects tools. However, it is not the purpose of this book to teach you EJB development. If you want to develop enterprise beans, you must already have a sound knowledge of the technology.

The book includes the following chapters:

- Chapter 2, "Introduction to EJB in WebObjects" (page 15), provides a brief introduction to Enterprise JavaBean technology and how it's implemented in WebObjects.
- Chapter 3, "Creating a Simple Session Bean" (page 19), walks you through the development of a simple session bean and its use in a bean-client application.
- Chapter 4, "Developing Bean Frameworks" (page 45), lists the steps you take to create and maintain bean frameworks.
- Chapter 5, "Configuring EJB Applications" (page 55), explains how to configure the transaction manager, the persistence manager, and the EJB container in your bean-client applications.
- Chapter 6, "Configuration Reference" (page 77), provides explanations of the XML tags used in the configuration files of bean-client applications.
- "Document Revision History" (page 123), lists changes made from previous editions of the book.

To get the most out of this book you should be an experienced WebObjects application developer. In particular, you need to know how to create applications using Project Builder and be familiar with the layout of a Project Builder project. To make use of enterprise beans in an application, you are required to edit configuration files written in XML; therefore, you should be familiar with XML's rules and syntax.

If you need to learn the basics about developing WebObjects applications, you can find that information in the following books:

- *Inside WebObjects: WebObjects Overview* provides you with a survey of WebObjects technologies and capabilities.
- *Inside WebObjects: Discovering WebObjects for HTML* shows you how to develop HTML-based applications.

- *Inside WebObjects: WebObjects Desktop Applications* shows you how to develop applications that leverage the power of desktop workstations as well as centralized servers and databases.
- *Inside WebObjects: Deploying WebObjects Applications* describes how to use WebObjects tools to deploy your applications as standalone entities.

For additional WebObjects documentation and links to other resources, visit http://developer.apple.com/webobjects.

If you need to learn about EJB development, these books provide you introductory information as well as development guidelines:

- *Enterprise JavaBeans* (O'Reilly)
- *Professional EJB* (Wrox Press)
- *Applying Enterprise JavaBeans: Component-Based Development for the J2EE Platform* (Addison-Wesley)

WebObjects uses open-source implementations of the EJB container, the object request broker, the transaction manager, and the persistence manager from Exolab (http://www.exolab.org). For details about those implementations in WebObjects, consult the following documents:

- *OpenEJB User Guide* provides details about the EJB container included with WebObjects. However, most of the information from it needed to develop or deploy enterprise beans is present in this book. You can find the document in `/System/Library/Frameworks/JavaOpenEJB.framework/Resources/English.lproj/Documentation/OpenEJB_User_Guide.pdf`.
- *OpenORB Programmers Guide* and *RMI over IIOP for OpenORB* deal with the Object Request Broker (ORB) implementation used in WebObjects. They are located in `/System/Library/Frameworks/JavaOpenORB.framework/Resources/English.lproj/Documentation`.
- API documentation on the Tyrex transaction manager is located in `/System/Library/Frameworks/JavaOpenTM.framework/Resources/English.lproj/Documentation`.

CHAPTER 1

About This Book

CHAPTER 2

Introduction to EJB in WebObjects

WebObjects provides all the tools you need to develop and deploy enterprise applications. However, WebObjects is not the only technology available. Other companies provide tools that accomplish the same task, albeit using different methods and requiring specialized deployment environments. Therefore, it's difficult for a WebObjects application to talk to an application developed and deployed under a different environment. J2EE and EJB bridge the schism between environments from different vendors.

J2EE standardizes the way Web applications communicate with the resources they need to operate. Akin to JDBC, the goal of J2EE is to provide an infrastructure that applications from different developers can utilize to get their work done.

Enterprise JavaBeans

Enterprise JavaBeans is an important part of J2EE. It provides an environment in which components from several manufacturers can be assembled into a working application. The application assembler, with deep knowledge of the requirements of the business, can choose the component that best matches the task at hand. For instance, she could use transaction-processing beans from one company; customer, order, and product beans from another company; and shipping beans from a third company. She would then end up with an application capable of accepting orders, charging the customer, and process shipments without having to write code.

Enterprise beans are specialized components that can encapsulate session information, workflow, and persistent data. A bean client is an application or an enterprise bean that makes use of another bean. An enterprise bean has three parts:

- **The home interface** is used by the client to create and discard beans.
- **The remote interface** is used by the client to execute the bean's business methods.
- **The implementation or bean class** is where the bean's business and callback methods are implemented. The client never invokes these methods directly; they are invoked by the bean container.

The container is a conceptual entity that mediates between enterprise-bean instances and client applications. Clients never access bean instances directly. Instead, they interact with proxies provided by the container This allows the bean container to perform its duties in the most efficient way. The client doesn't have to know how the container implements its functions; all it needs to know is how to talk to the container.

In addition, beans have a deployment descriptor. This is an XML file that gives the container information about each bean and data-source connection details, among many other items.

There are two major types of enterprise beans: session beans and entity beans.

- **Session beans** come in two flavors: stateful and stateless. Stateful session beans maintain state between method invokations; stateless session beans do not.

  Stateless session beans are useful for grouping related methods in one place. Stateful session beans can be used to encapsulate workflow. In most cases it's more efficient for client applications to use session beans (stateless or stateful) to accomplish their tasks than to use entity beans directly because network traffic is reduced.

- **Entity beans** are similar to enterprise objects in WebObjects's Enterprise Object technology. They encapsulate access to data entities.

An enterprise-bean developer can focus on the high-level business logic needed to implement the services that a bean provides instead of on low-level system or data-store calls (those functions can be left to the container).

One of the most important functions of the container is bean pooling, which is used to share bean instances among several clients. WebObjects includes the OpenEJB open-source container system. OpenEJB consists of four main components:

- **EJB container:** The EJB container implements the lifecycle of enterprise beans and the server contracts in the EJB specification.
- **Object Request Broker (ORB):** The OpenORB object request broker implements RMI-over-IIOP, naming service, and CORBA ORB.
- **Transaction manager:** The Tyrex transaction manager implements the Java Naming and Directory Interface (JNDI) specification, and a transaction manager compliant with the Java Transactions (JTA) and Object Transaction Service (OTS) specifications.
- **Persistence manager:** The Castor JDO persistence manager implements bean persistence for entity beans. It's used in the implementation of CMP (container-managed persistence) beans. An entity bean is a type of enterprise bean that represents a data entity, such as a Person or a Department.

Enterprise JavaBeans in WebObjects

You can use WebObjects development tools to develop enterprise beans from scratch or to integrate third-party EJB-based solutions in a WebObjects application. Bean development in WebObjects is divided in two phases: development and deployment.

You develop enterprise beans by writing the `.java` and deployment descriptor files. Project Builder provides you with templates for all these files. You can also obtain the source code or JAR files for enterprise beans from a third party.

You deploy one or more beans by generating a bean framework, which contains the beans' JAR and deployment descriptor files, and placing it somewhere in a development computer's file system; for example, in `/Library/Frameworks`. After you deploy a bean framework, it's available to be integrated in client applications for their use.

Client applications can be developed in two ways: using an internal bean container, or using an external container:

- **Internal container:** This approach is the most scalable because each application instance has its own container and naming-service object.

  If the user load of your site becomes too large for one instance to handle, all you have to do is add more instances of it. Each container answers only to one application, so there is no application-to-container bottleneck.

- **External container:** This approach is beneficial if you already have a robust bean container, running on a fast computer, that you want to leverage. In this case, no configuration files should be present in the bean-client application project. For more on the configuration files, see "Configuring EJB Applications" (page 55).

Creating a Simple Session Bean

Before developing WebObjects applications that use enterprise beans, you have to create enterprise-bean frameworks. These frameworks contain the JAR and deployment descriptor files needed to deploy enterprise beans.

You can create a bean framework by writing the beans yourself or by using third-party beans (either from Java source and deployment descriptor files or JAR files). Project Builder helps you create beans from scratch by providing you with bean templates that get you started.

In most cases, you save both time and money when you use third-party beans in your projects instead of developing them from scratch. This is because you obtain a solution that has been tested by the solution vendor and other developers like you. Also remember that you cannot take advantage of Enterprise Object technology in your enterprise beans; for example, you have to implement primary-key classes, finder methods, primary-key–value generation, and so on in your entity beans. In addition, you have to choose between implementing a bean as an entity bean or a session bean. It's a bean provider's job to design an effective and efficient bean solution for you. You can then compare similar solutions from various vendors and purchase the one that most closely addresses your situation.

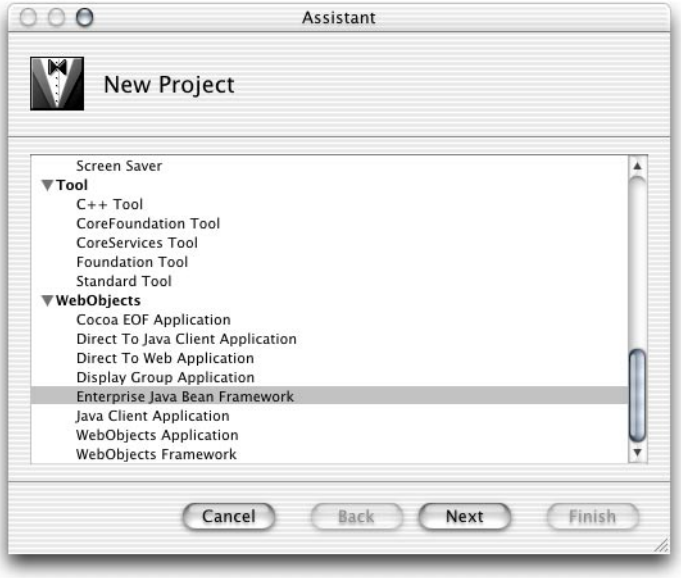
The following sections show you how to develop a stateless session bean for use in a WebObjects application both on Mac OS X and Windows.

Developing an Enterprise Bean in Mac OS X

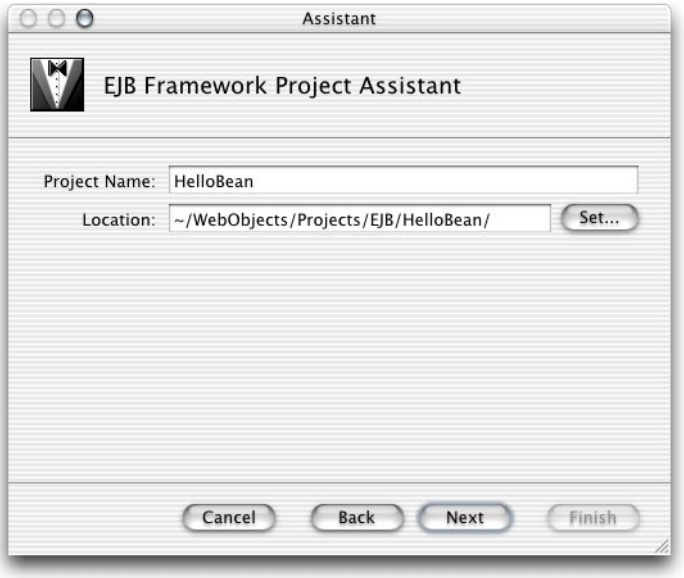
This section shows you how you develop a stateless session bean for use in a WebObjects application in Mac OS X.

Creating the Bean Framework

1. Launch Project Builder.
2. Choose File > New Project.
3. In the New Project pane of the Project Builder Assistant, select Enterprise Java Bean Framework from the list of project types, and click Next.

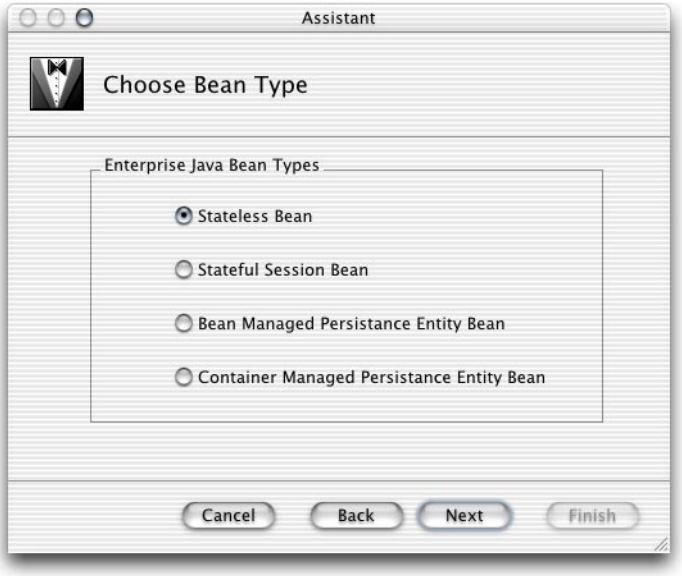


4. In the EJB Framework Project Assistant pane of the Assistant:
   a. Enter `HelloBean` in the Project Name text field.
   b. Enter a location for the project in the Location text field.
   c. Click Next.



5. In the Create New Enterprise JavaBean pane of the Assistant, select “Create source files for a new Enterprise JavaBean” and click Next.

6. In the Choose Bean Type pane, make sure Stateless Bean is selected under Enterprise Java Bean Types, and click Next.

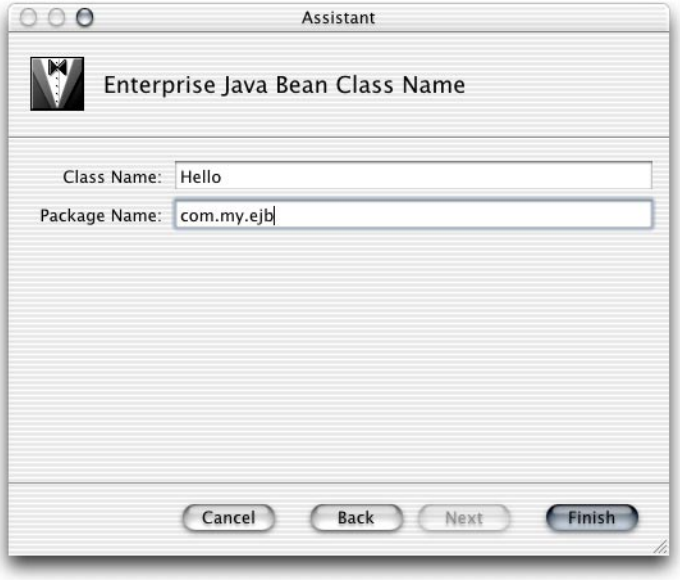


7. In the Enterprise JavaBean Class Name pane:

    a. Enter `Hello` in the Class Name text field.

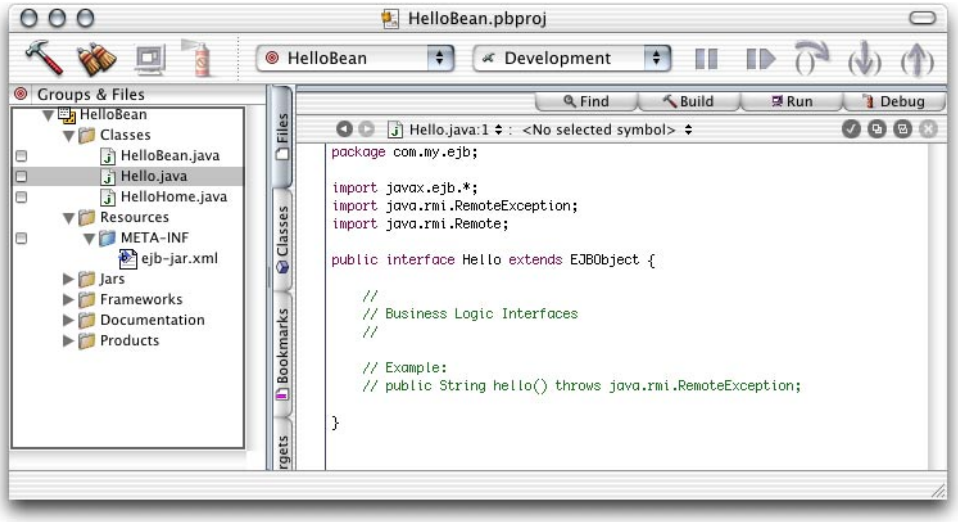
    b. Enter `com.my.ejb` in the Package Name text field.

    c. Click Finish.



After you're done, you'll see a window similar to the one in Figure 3-1.

Figure 3-1 HelloBean framework project



Analyzing the Hello Bean's Files

The HelloBean project has templates for the home and remote interfaces, as well as for the implementation class of the Hello enterprise bean in the Classes group of the Groups & Files list. In addition, the Resources group contains the bean's deployment descriptor.

This is the template for the home interface of the Hello enterprise bean (`HelloHome.java`):

```
package com.my.ejb;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface HelloHome extends EJBHome {

    /** Creation methods **/
```

```
    /* Stateful session beans may have multiple create methods taking
    * different parameters. They must all be reflected in identically
    * named methods in the home interface without the 'ejb' prefix
    * and initial cap.
    *
    * Stateless session bean create methods never have parameters.
    */

    public Hello create() throws RemoteException, CreateException;
}
```

This is the template for the bean's remote interface (`Hello.java`):

```
package com.my.ejb;
import javax.ejb.*;
import java.rmi.RemoteException;
import java.rmi.Remote;

public interface Hello extends EJBObject {

    //
    // Business Logic Interfaces
    //

    // Example:
    // public String hello() throws java.rmi.RemoteException;

}
```

This is the template for the bean's implementation class (`HelloBean.java`):

```
package com.my.ejb;
import javax.ejb.*;

public class HelloBean implements SessionBean {

    //
    // Creation methods
    //

    public HelloBean() {
```

```
    }

    public void ejbCreate() throws CreateException {
        /* Stateless session bean create methods never have parameters */
    }

    //
    // SessionBean interface implementation
    //

    private SessionContext _ctx;

    public void setSessionContext(SessionContext ctx) {
         this._ctx = ctx;
    }

    public void ejbPassivate() {
        /* does not apply to stateless session beans */
    }

    public void ejbActivate() {
        /* does not apply to stateless session beans */
    }

    public void ejbRemove() {
        /* does not apply to stateless session beans */
    }

    //
    // Business Logic Implementations
    //

    // Example:
    // public String hello() { return "hello"; }
}
```

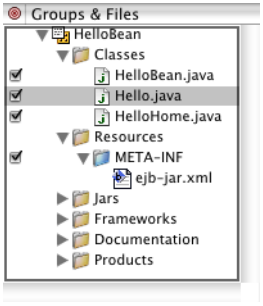
This is the bean's deployment descriptor (`ejb-jar.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
```

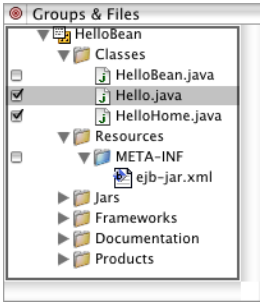
```
<ejb-jar>
  <description>deployment descriptor for HelloBean</description>
  <display-name>HelloBean</display-name>
  <enterprise-beans>
        <session>
            <description>deployment descriptor for HelloBean</description>
            <display-name>HelloBean</display-name>
            <ejb-name>HelloBean</ejb-name>
            <home>com.my.ejb.HelloHome</home>
            <remote>com.my.ejb.Hello</remote>
            <ejb-class>com.my.ejb.HelloBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
</ejb-jar>
```

In addition to providing you with most of the code needed to deploy a bean, Project Builder also partitions the source code appropriately between two targets: EJB Deployment and EJB Client Interfaces.

If you choose the EJB Deployment target, you'll see that all of the bean's source files are assigned to it.



When you view the EJB Client Interfaces target, however, you see that the implementation class and the deployment descriptor files are not assigned to it.



Adding Business Logic to the Bean

Now you're ready to add the business logic required for the Hello bean to provide a message to its clients.

Edit `Hello.java` by adding the following code:

```
public String message() throws RemoteException;
```

Edit `HelloBean.java` by adding the implementation of the `message` method, which is listed below.

```
public String message() {
    return "Hello World!";
}
```

Building the Bean Framework

To build the HelloBean framework, all you have to do is click the build icon or choose Build > Build. (Make sure that the HelloBean target is selected in the target pop-up menu before you build.)

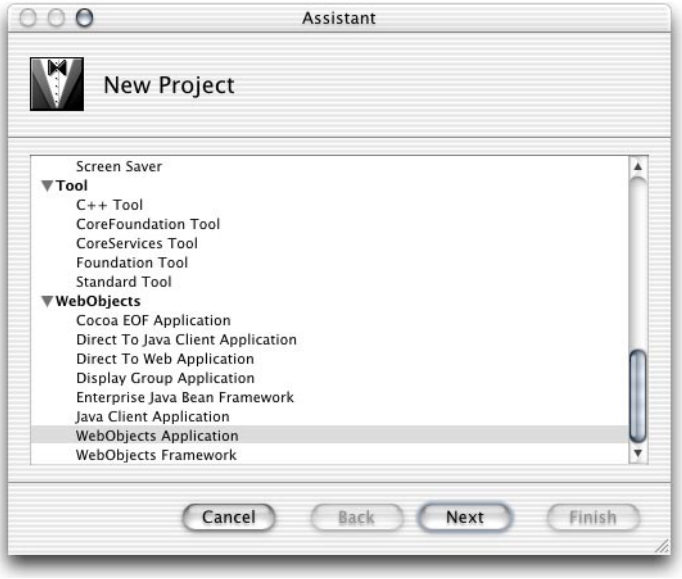
After the framework is built, you can find it in the project's `build` directory:

```
HelloBean/
    build/
        HelloBean.framework
```

Creating the Client Application

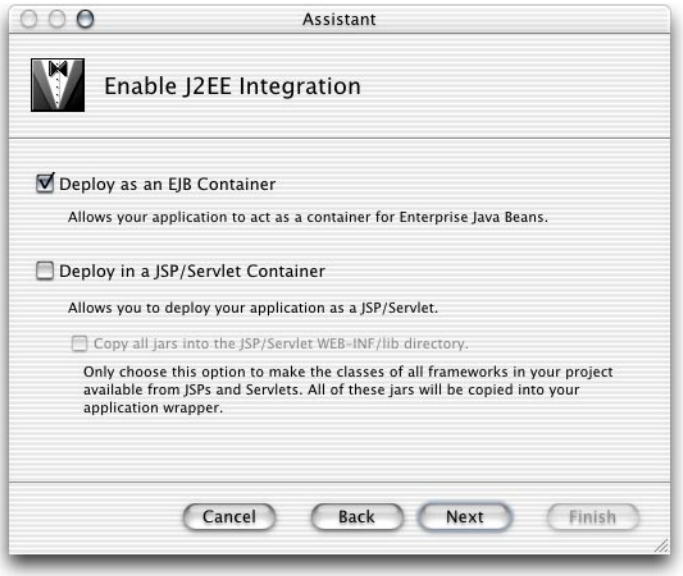
Now that the HelloBean framework is built, you're ready to use it in an application. In this case, the client application is an HTML-based application that invokes the bean's message method, and displays its return value in a WOString element.

1. Create a WebObjects Application project.



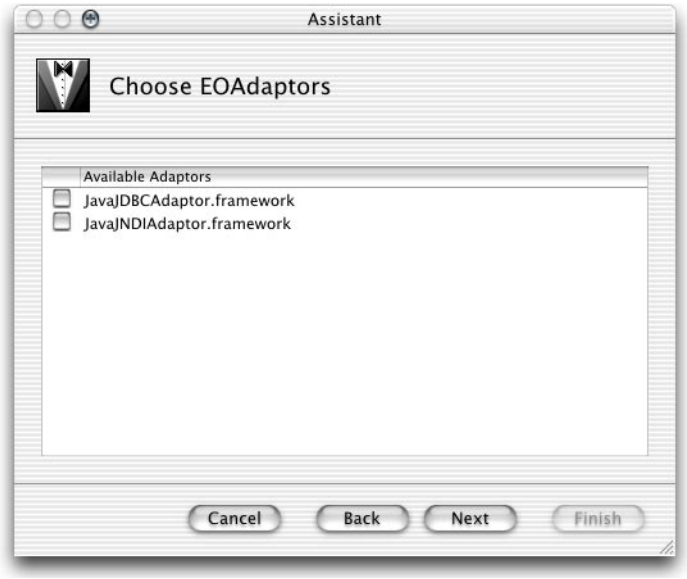
2. Name the project HelloBean_Client, and click Next.

3. In the Enable J2EE Integration pane of the Project Builder Assistant, select "Deploy as an EJB Container" and click Next.



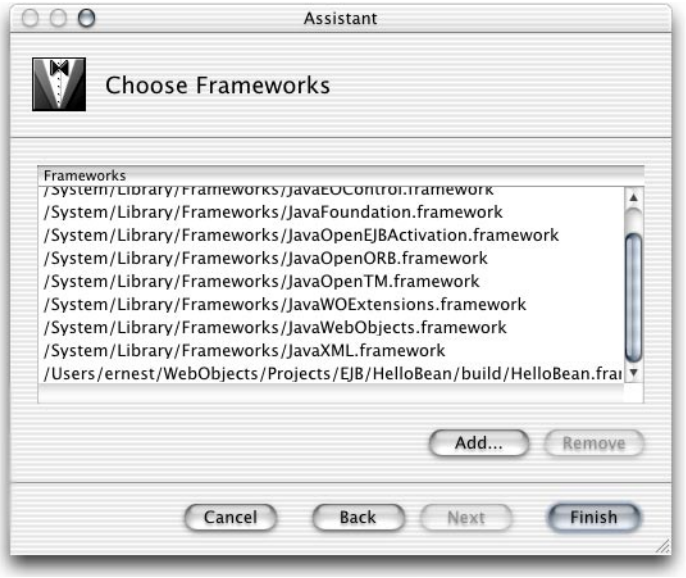
When you deploy the client application as an EJB container, each application instance has its own EJB container. See "Enterprise JavaBeans in WebObjects" (page 17) for details in internal and external containers.

4. This example doesn't require the use of any data-source adaptors, so make sure no adaptors are selected in the Choose EOAdaptors pane.



You need to select a data-source adaptor only if your application uses enterprise objects in addition to enterprise beans. Entity beans that use bean-managed persistence (BMP) are responsible for interfacing with the necessary data stores. For entity beans that use container-managed persistence (CMP), the bean container has this responsibility. This example application does not use enterprise objects.

5. Add the HelloBean framework to the project.
   a. In the Choose Frameworks pane of the Assistant, click Add.



   b. Select `HelloBean.framework` in the build folder of the HelloBean project folder, and click Choose.
6. Click Finish.

Adding Business Logic to the Bean Client

You have generated a WebObjects application that, when run, instantiates its own EJB container. This container behaves like a standard EJB container. To access bean instances, you use standard EJB methods.

Editing Session.java

Now, you'll edit `Session.java` so that each new session creates a Hello-bean proxy that your components can access.

First, add these `import` statements:

```
import com.my.ejb.Hello;
import com.my.ejb.HelloHome;
import java.rmi.RemoteException;
import java.util.Properties;
import javax.ejb.CreateException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
```

Now, add two instance variables: one to hold the Hello bean's home interface and another to hold its remote interface.

```
// holds HelloBean's home interface
protected Hello hello;

// holds HelloBean's remote interface
private HelloHome _helloHome = null;
```

Modify the `Session` constructor so that it looks like this:

```
public Session() {
    super();

    // instantiate a HelloBean object
    try {
        hello = helloHome().create();

    } catch (RemoteException re) {
        re.printStackTrace();
    } catch (CreateException ce) {
        ce.printStackTrace();
    }
}
```

Finally, add the following method:

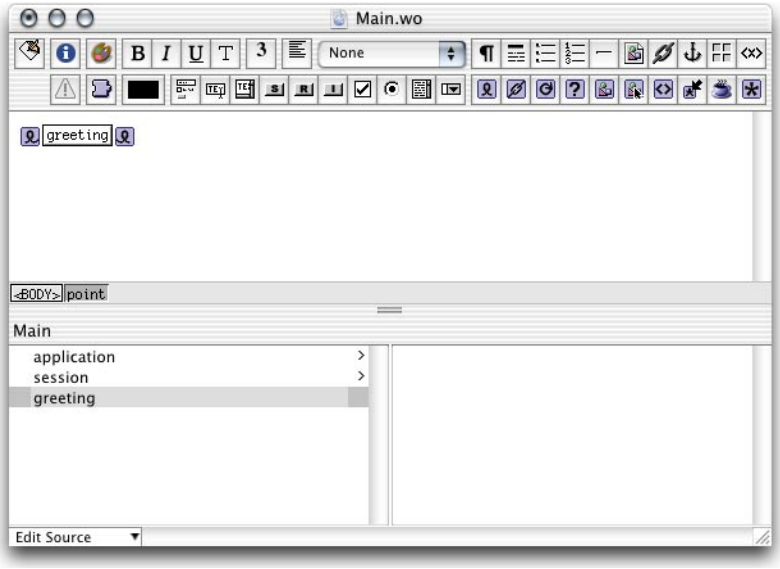
```
// gets HelloBean's home interface
public HelloHome helloHome() {
    if (_helloHome == null) {
        try {
            Context jndiContext = new InitialContext();
            _helloHome =
(HelloHome)PortableRemoteObject.narrow(jndiContext.lookup("HelloBean"),
HelloHome.class);

        } catch (NamingException ne) {
            ne.printStackTrace();
        }
    }

    return _helloHome;
}
```

Editing Main.wo

Open `Main.wo` in WebObjects Builder by double-clicking `Main.wo`, which is located under the Main subgroup of the Web Components group in the Groups & Files list.

Add a String key called `greeting` to `Main.wo` through the Edit Source pop-up menu.

Add a WOString element to the component, and bind it to the `greeting` key.



Editing Main.java

When a Main page is about to be displayed, the Main object needs to invoke the `message` method of its Hello bean proxy to obtain the bean's greeting, and store the value returned in its `greeting` instance variable. When the WOString element is rendered on the page, its `value` binding provides the text to be displayed; in this case, the value comes from `greeting` in the Main object.

Add the following `import` statements to `Main.java`:

```
import com.my.ejb.Hello;
import java.rmi.RemoteException;
```

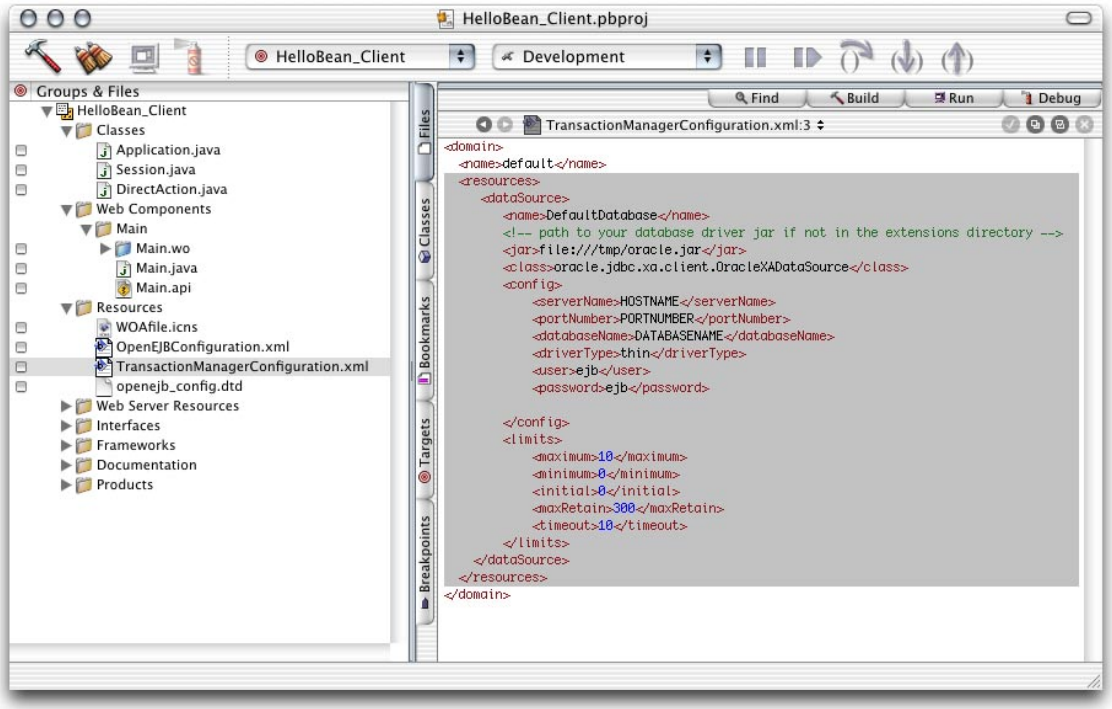
Edit the `Main` constructor so that it looks like this:

```
public Main(WOContext context) {
    super(context);

    Session session = (Session)session();

    try {
        greeting = session.hello.message();

    } catch (RemoteException re) {
        re.printStackTrace();
    }
}
```

Configuring the Container

This simple session bean project doesn't make use of bean persistence. Therefore, it requires no container configuration. Figure 3-2 (page 38) shows the text you need to delete from the `TransactionManagerConfiguration.xml` file of the project (everything between the `<resources>` and `</resources>` tags and the tags themselves).

Figure 3-2 Extraneous transaction manager configuration information (for applications without CMP beans)



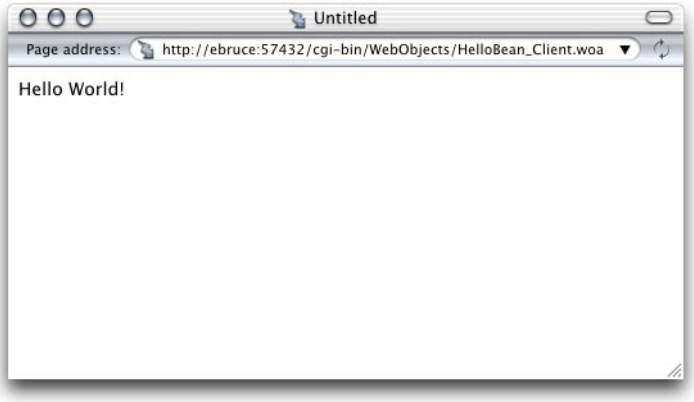
After removing the irrelevant information, the `TransactionManagerConfiguration.xml` file should look like this:

```
<domain>
  <name>default</name>
</domain>
```

Running the HelloBean_Client Application

After you build and run the application, you should see a window similar to the one in Figure 3-3 (page 39) in your Web browser.

Figure 3-3 Output of the HelloBean_Client application



Developing an Enterprise Bean in Windows

This section shows you how you develop a stateless session bean for use in a WebObjects application in Windows.

Creating the Bean Framework

1. Launch Project Builder.
2. Choose Project > New.
3. Choose Java WebObjects EJB Framework from the Project Type pop-up menu in the New Project dialog, and click Browse.
4. Select a path for your project, name it HelloBean, and click Save.
5. In the Specify Enterprise JavaBeans pane of the EJB Framework Wizard, select "Create source files for a new Enterprise Java Bean" and click Next.
6. Make sure that Stateless Session Bean is selected in the Chose Enterprise JavaBeans Type pane of the wizard and click Next.

7. In the Create New Enterprise JavaBeans class pane:
    a. Enter `Hello` in the Class Name text field.
    b. Enter `com.my.ejb` in the Package Name text field.
    c. Click Finish.

Adding Business Logic to the Bean

Now you're ready to add the business logic required for the Hello bean to provide a message to its clients.

Edit `Hello.java` by adding the following code (the file is located in the Classes bucket):

```
public String message() throws RemoteException;
```

Edit `HelloBean.java` by adding the implementation of the `message` method, which is listed below (the file is located in the Classes bucket of the EJBServer subproject).

```
public String message() {
    return "Hello World!";
}
```

Building the Framework

To build the HelloBean framework, click the Build button or choose Tools > Project Build > Build.

After the framework is built, you'll find it in the project's directory:

```
HelloBean/
    HelloBean.framework
```

Creating the Client Application Project

Now that the HelloBean framework is built, you're ready to use it in an application. In this case, the client application is an HTML-based application that invokes the bean's message method, and displays its return value in a WOString element.

1. Create a Java WebObjects Application project and name it HelloBean_Client.
2. Choose None in the "Choose type of assistance in your Java project" pane of the WebObjects Application Wizard.
3. Choose "Deploy as an EJB Container" in the Enable J2EE Integration pane.
4. In the Choose EOAdaptors pane, click Select None, and then click Finish.

Adding the HelloBean Framework to the HelloBean_Client Project

You need to add the HelloBean framework to the HelloBean_Client project in order to use the services provided the Hello enterprise bean—mainly providing a greeting. To accomplish that, follow these steps:

1. Select the Frameworks bucket and choose Project > Add Files.
2. Navigate to the HelloBean project directory, select `HelloBean.framework`, and click Open.
3. Click Add in the search order dialog.

Creating the Container Configuration Files

To create the configuration files that the client application needs to interact with its environment, you need to run an application named OpenEJBTool, whose launch script is located in `/Apple/Library/WebObjects/JavaApplications/OpenEIBTool.woa`.

Using the Bourne shell, execute the following commands:

```
cd /Apple/Library/WebObjects/JavaApplications/OpenEJBTool.woa

./OpenEJBTool.cmd -o c:/<HelloBean_Client_path>
                  c:/<HelloBean_path>HelloBean.framework
```

When the tool is finished, you need to add the configuration files it generated (`OpenEJBConfiguration.xml` and `TransactionManagerConfiguration.xml`) to the Resources bucket of the HelloBean_Client project.

Note: You have to run OpenEJBTool manually every time you add bean frameworks to your project.

Adding Business Logic to the Bean Client

You have generated a WebObjects application that, when run, instantiates its own EJB container. This container behaves like a standard EJB container. To access bean instances, you use standard EJB methods.

Editing Session.java

Now, you'll edit `Session.java` so that each new session creates a Hello proxy and provides access to it to components.

First, add these `import` statements:

```
import com.my.ejb.Hello;
import com.my.ejb.HelloHome;
import java.rmi.RemoteException;
import java.util.Properties;
import javax.ejb.CreateException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
```

Now, add two instance variables: one to hold the Hello bean's home interface and another to hold its remote interface.

```
// holds HelloBean's home interface
protected Hello hello;

// holds HelloBean's remote interface
private HelloHome _helloHome;
```

Modify the `Session` constructor so that it looks like this:

```
public Session() {
    super();

    // instantiate a HelloBean object
    try {
        hello = helloHome().create();

    } catch (RemoteException re) {
```

```
        re.printStackTrace();
    } catch (CreateException ce) {
        ce.printStackTrace();
    }
}
```

Finally, add the following method:

```
// gets HelloBean's home interface
public HelloHome helloHome() {
    if (_helloHome == null) {
        try {
            Context jndiContext = new InitialContext();
            _helloHome =
(HelloHome)PortableRemoteObject.narrow(jndiContext.lookup("HelloBean"),
HelloHome.class);

        } catch (NamingException ne) {
            ne.printStackTrace();
        }
    }

    return _helloHome;
}
```

Editing Main.wo

Open `Main.wo` in WebObjects Builder by double-clicking `Main.wo`, which is located under the Web Components bucket.

Add a String key called `greeting` to `Main.wo` through the Edit Source pop-up menu.

Add a WOString element to the component, and bind it to the `greeting` key.

Editing Main.java

When a Main page is about to be displayed, the Main object needs to invoke the `message` method of its Hello bean proxy to obtain the bean's greeting, and store the value returned in its `greeting` instance variable. When the WOString element is rendered on the page, its `value` binding provides the text to be displayed; in this case, the value comes from `greeting` in the Main object.

Add the following `import` statements to `Main.java`:

```
import com.my.ejb.Hello;
import java.rmi.RemoteException;
```

Edit the `Main` constructor so that it looks like this:

```
public Main(WOContext context) {
    super(context);

    Session session = (Session)session();

    try {
        greeting = session.hello.message();

    } catch (RemoteException re) {
        re.printStackTrace();
    }
}
```

Configuring the Container

This simple session bean project doesn't make use of bean persistence. Therefore, it requires no container configuration. You need to edit the `TransactionManagerConfiguration.xml` file of the project to remove extraneous data-source configuration information, which is everything between the `<resources>` and `</resources>` tags, and the tags themselves.

After removing the irrelevant information, the `TransactionManagerConfiguration.xml` file should look like this:

```
<domain>
  <name>default</name>
</domain>
```

Running the Hello_Client Application

After you build and run the application, you should see a Web browser window with the message "Hello World!"

CHAPTER 4

Developing Bean Frameworks

This chapter tells you how to create enterprise-bean framework to be used by client applications. It contains the following sections:

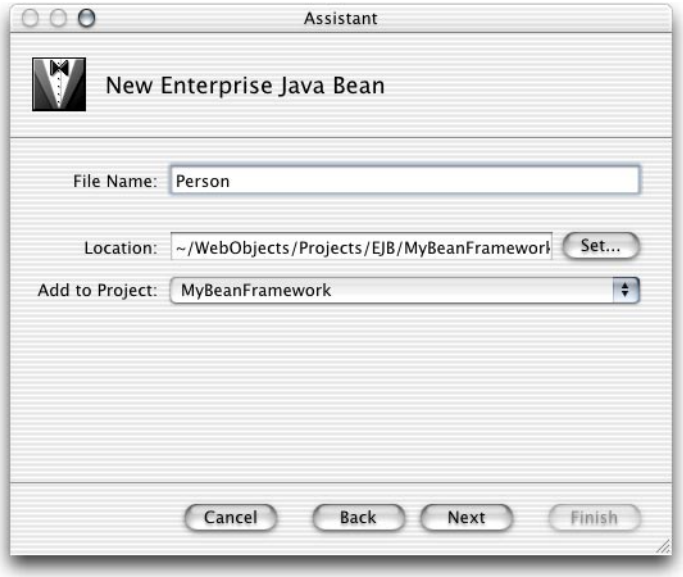
- "Adding Enterprise-Bean Source Files to a Bean- Framework Project" (page 45).
- "Adding Enterprise-Bean JAR Files to a Bean Framework Project" (page 48).
- "Creating Enterprise-Bean Frameworks From Bean JAR Files in Windows" (page 50).
- "Adding CMP Fields to an Enterprise Bean Deployment Descriptor" (page 53).

Adding Enterprise-Bean Source Files to a Bean-Framework Project

To add new enterprise-bean source files to an existing enterprise-bean framework project, follow these steps:

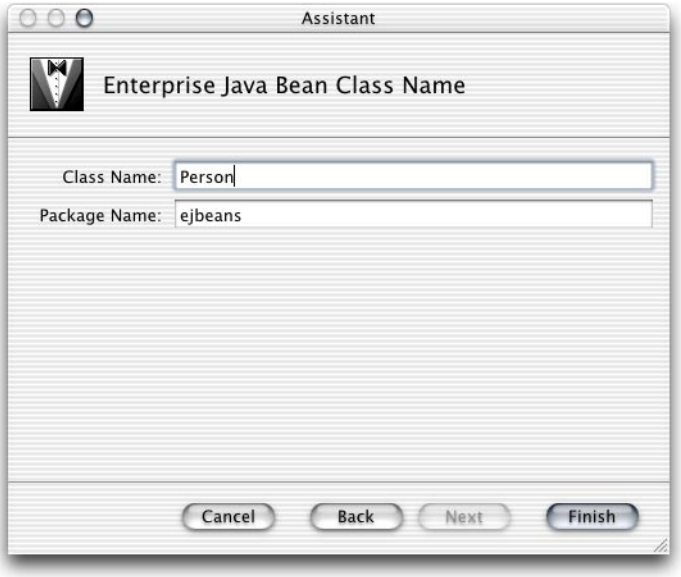
1. Choose File > New File.
2. Choose Enterprise Java Bean from the New File pane of the Project Builder Assistant.

3. In the New Enterprise Java Bean pane of the Assistant:
   a. Enter the name of the bean in the File Name text field.
   b. Enter the location where you want to place the bean's source files in the Location text field.
   c. Choose the project you want to add the bean to from the Add to Project pop-up menu.
   d. Click Next.



4. Select “Create source files for a new Enterprise Java Bean” in the Create New Enterprise Java Bean pane and click Next.

5. In the Enterprise Java Bean Class Name pane:
    a. Enter the class name of the bean in the Class Name text field.
    b. Enter the package name in the Package Name text field.
    c. Click Finish.



Adding Enterprise-Bean JAR Files to a Bean Framework Project

To add an enterprise-bean JAR file to an existing enterprise-bean framework project follow these steps:

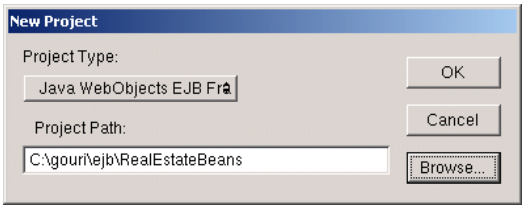
1. Choose File > New File.
2. Choose Enterprise Java Bean form the New File pane of the Project Builder Assistant.

3. In the New Enterprise Java Bean pane of the Assistant
   a. enter the name of the bean in the File Name text field
   b. choose the project you want to add the bean to from the Add to Project pop-up menu
   c. click Next
4. In the Create New Enterprise Java Bean pane
   a. select Use JAR Files
   b. enter the location of the JAR files you want to add in the Client Interfaces JAR and Deployment JAR text fields (client-interface JAR files contain helper classes that facilitate communication between clients and bean containers)

Creating Enterprise-Bean Frameworks From Bean JAR Files in Windows

In Windows, you have to create one enterprise-bean framework per JAR file. Follow these steps to create an enterprise-bean framework:

1. Launch Project Builder.
2. Choose Project > New.
3. Choose Java WebObjects EJB Framework from the Project Type pop-up menu in the New Project dialog and enter a location for your project. Then click OK.

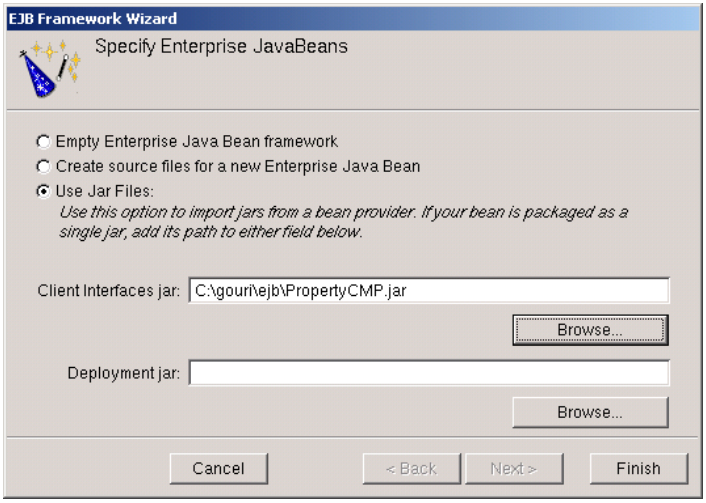


4. In the Specify Enterprise JavaBeans pane of the EJB Framework Wizard:

   a. Select "Use JAR Files."

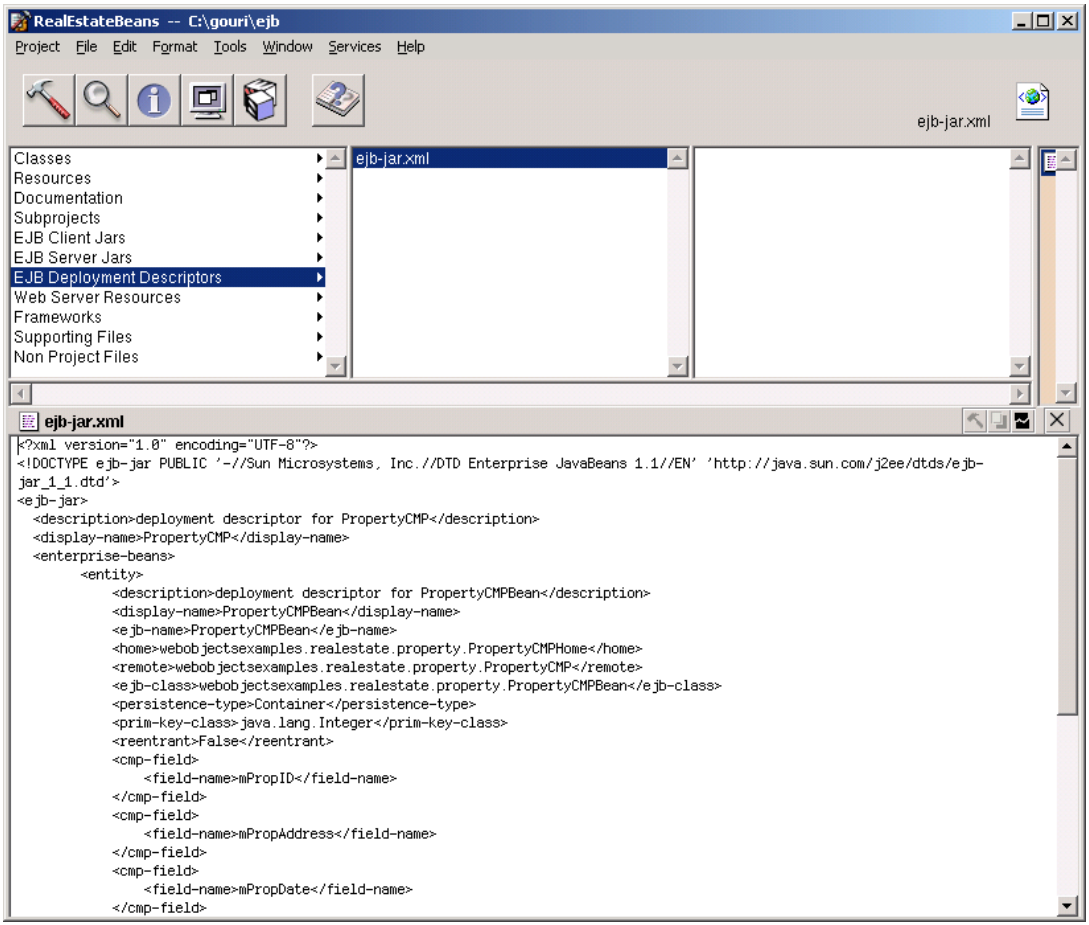
   b. Enter the location of the JAR file you want to use in the "Client Interfaces jar" text field.

   c. Click Finish.



After WebObjects finishes generating the project, you should see a window like the one in Figure 4-1.

Figure 4-1 Bean framework project in Windows



Adding CMP Fields to an Enterprise Bean Deployment Descriptor

After creating a bean framework using Project Builder, you have to add to the deployment descriptor the fields whose persistence is to be managed by the EJB container. To accomplish this, you add `<cmp-field>` and `</cmp-field>` tags to the `ejb-jar.xml` file in the META-INF directory of your project.

Listing 4-1 lists the deployment descriptor of a simple entity bean with container-managed persistence.

Listing 4-1 Deployment descriptor for a CMP entity bean

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <description>deployment descriptor for _test90</description>
  <display-name>_test90</display-name>
  <enterprise-beans>
        <entity>
            <description>deployment descriptor for PersonBean</description>
            <display-name>PersonBean</display-name>
            <ejb-name>PersonBean</ejb-name>
            <home>ejbeans.PersonHome</home>
            <remote>ejbeans.Person</remote>
            <ejb-class>ejbeans.PersonBean</ejb-class>
            <persistence-type>Container</persistence-type>
            <prim-key-class>ejbeans.PersonPK</prim-key-class>
            <reentrant>False</reentrant>
            <resource-ref>
             <description>the default datasource for a CMP bean.</description>
                <res-ref-name>jdbc/DefaultCMPDatasource</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
```

```
                <res-auth>Container</res-auth>
            </resource-ref>
            <cmp-field>
              <field-name>personID</field-name>
            </cmp-field>
            <cmp-field>
              <field-name>firstName</field-name>
            </cmp-field>
            <cmp-field>
              <field-name>lastName</field-name>
            </cmp-field>
        </entity>
    </enterprise-beans>
</ejb-jar>
```

Configuring EJB Applications

This chapter shows you how to configure a bean-client application project so that it can use resources like databases and JavaMail connections, and it explains how to ensure that each enterprise bean is bound to the appropriate resources.

You need to configure three major items before deploying a bean-client application:

- **Transaction manager.** This is where you define the data stores that your enterprise beans use to store their data and the JavaMail connections they use for messaging.
- **Persistence manager.** Here you map the fields of your CMP (container-managed persistence) beans to columns in tables of your data stores so that the container can perform database transactions for the beans.
- **EJB container.** This is where you set bean-deployment properties such as method transactions and permissions.

You perform this configuration by editing the files in Table 5-1.

Table 5-1 The configuration files of a bean-client application

Filename	Purpose
`TransactionManagerConfiguration.xml`	Defines data stores and JavaMail connections.
`LocalTransactionConfiguration.xml`	Defines data-store connection information for local databases.

Table 5-1 The configuration files of a bean-client application

Filename	Purpose
`GlobalTransactionConfiguration.xml`	Defines the JNDI name of a remote data store.
`CMPConfiguration.xml`	Defines bean-to-table and field-to-column mapping.
`OpenEJBConfiguration.xml`	Defines enterprise-bean deployment behavior for the container.

The chapter is divided in the following sections:

- "Configuration Overview" (page 56) provides you with a checklist of items you need to review in the configuration files that WebObjects creates by default.
- "Transaction Manager Configuration" (page 61) explains how to configure the transaction manager.
- "Persistence Manager Configuration" (page 62) tells you how to map enterprise-bean fields to table columns.
- "Container Configuration" (page 70) explains how you configure the EJB container.
- "Using External Containers" (page 73) shows you how to configure your client application to use an external EJB container.

Configuration Overview

This section gives you a quick look at the configuration process for bean-client applications. It lists the major points you need to look at before deploying your application. It's divided in the following sections:

- "Configuring the Transaction Manager" (page 57).
- "Configuring the EJB Container" (page 58).
- "Configuring the Persistence Manager" (page 60).

Configuring the Transaction Manager

The TransactionManagerConfiguration.xml file is where you enter connection information, such as user name and password, for the data stores that your application's CMP beans use. You also configure JavaMail. The OpenEJBConfiguration.xml file determines what you need to configure in this file: the data stores, or JavaMail.

If your enterprise beans use container-managed persistence (the OpenEJBConfiguration.xml file contains the string "<res-type>javax.sql.DataSource</res-type>"), you need to configure at least one data store.

```
<dataSource>
    <name>DefaultDatabase</name>
    <!-- path to your database-driver JAR file if not present in the
extensions directory -->
    <jar>file:///tmp/oracle.jar</jar>
    <class>oracle.jdbc.xa.client.OracleXADataSource</class>
    <config>
        <serverName>gamow</serverName>
        <portNumber>1968</portNumber>
        <databaseName>bigbang</databaseName>
        <driverType>thin</driverType>
        <user>george</user>
        <password>fire</password>
    </config>
    <limits>
        <maximum>10</maximum>
        <minimum>0</minimum>
        <initial>0</initial>
        <maxRetain>300</maxRetain>
        <timeout>10</timeout>
    </limits>
</dataSource>
```

If your enterprise beans do not use container-managed persistence, you need to delete the resources section of the TransactionManagerConfiguration.xml file, which includes everything between the <resources> and </resources> tags as well as the tags themselves.

If you need to define more than one data store, you can add <dataSource> tags for each additional data store. See "Transaction Manager Configuration" (page 61) for more information.

If your enterprise beans make use of JavaMail (the OpenEJBConfiguration.xml file contains the string "<resource-type>javax.mail.Session</resource-type>"), you need to configure JavaMail.

To configure JavaMail in the TransactionManagerConfiguration.xml file, add this to the data-source section and customize as necessary:

```
<javamail>
    <name>DefaultSMTPServer</name>
    <property>
        <key>mail.smtp.host</key>
        <value>post.office.com</value>
    </property>
</javamail>
```

Configuring the EJB Container

Once you have defined the resources that your enterprise beans utilize, you have to review the container environment that WebObjects has defined for you in the OpenEJBConfiguration.xml file:

- Data sources and JavaMail-connection factories.

  If your enterprise beans use more than one data source or rely on JavaMail (as defined in the TransactionManagerConfiguration.xml file, you have to make sure that each bean is linked to the appropriate data source or JavaMail connection factory (through the <res-id> tag inside <resource-ref>) in the OpenEJBConfiguration.xml file. See "<resource-ref>" (page 115).

- Environment entries.

  Scan the file for <env-entry> tags and make sure that they contain the appropriate values for your situation. See "<env-entry>" (page 107).

- Method-transaction settings.

  Make sure that the <trans-attribute> of <method-transaction> tags is set to the appropriate transaction type. If the enterprise bean does not define the transaction type for a method, WebObjects sets it to Required. See "Containers Section" (page 70), and "<method-transaction>" (page 112) for details.

- Security-role to physical-role mapping.

  By default, all methods are assigned a default, liberal logical security role. For details, see "<method-permission>" (page 112).

```
<method-permission>
    <description>Default liberal method permission</description>
    <role-name>Default liberal logical role</role-name>
    <method>
        <ejb-deployment-id>AgentBMPBean</ejb-deployment-id>
        <method-name>*</method-name>
    </method>
</method-permission>
```

- Performance.

  If you need to tailor the performance of your application, you can fine-tune the container by adjusting the pool size, the eviction or passivation strategy, and so on. For details see "Configuration Reference" (page 77).

- One `<entity-container>` tag per database.

  When your CMP beans use more than one database, you need to

  - group the CMP beans that use the same data store under the same `<entity-container>` tag
  - create a local transaction manager configuration file by duplicating the `LocalTransactionConfiguration.xml` file and changing `"Local_TX_Database"` so that it names the additional data store (for example, `"Local_TX_Personnel"`)
  - create a global transaction manager configuration file by duplicating the `GlobalTransactionConfiguration.xml` file and changing `"Global_TX_Database"` so that it names the additional data store (for example, `"Global_TX_Personnel"`)

  In general, you should use the following grouping:

  - one `<entity-container>` tag per distinct database that encloses its corresponding BMP beans (for more information, see "<entity-container>" (page 107))
  - one `<stateless-session-container>` tag that encloses all stateless beans
  - one `<stateful-session-container>` tag that encloses all stateful beans

Configuring the Persistence Manager

This section explains how to configure the container for CMP beans. If your application doesn't use CMP beans, you don't need to configure the files mentioned here. In fact, the files are only present in your project when at least one of your enterprise beans uses container-managed persistence.

LocalTransactionConfiguration.xml

You define a data-store connection for the container in XML files. WebObjects provides you with a default configuration in the `LocalTransactionConfiguration.xml` file. If your enterprise beans use additional data stores, you need to create additional files, one per data store. You can duplicate the default one and add it to your project. For details on how to configure databases, see "Elements of the Data-Store Configuration Files" (page 90).

GlobalTransactionConfiguration.xml

This is where you define the JNDI name of a remote data store. It must be identical to the name used in the `<resource-ref>` tag of a bean in the `CMPConfiguration.xml` file. As with the `LocalTransactionConfiguration.xml` file, you must have one per data store. For more information on local and global data-store configuration files, see "<database>" (page 92).

CMPConfiguration.xml

This is where you map enterprise beans to tables and their fields (or instance variables) to columns in those tables. You also define a bean's identity or primary key and configure key-value generators. This is an example of a `CMPConfiguration.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapping PUBLIC "-//EXOLAB/Castor Mapping DTD Version 1.0//EN"
                         "http://castor.exolab.org/mapping.dtd">
<mapping>
    <class key-generator="MAX" identity="mPropID"
name="webobjectsexamples.realestate.property.PropertyCMPBean">
        <map-to table="EJB_PROPERTY"/>
        <field direct="true" name="mPropID" type="java.lang.Integer">
            <sql name="PROP_ID" type="integer"/>
        </field>
```

```
        <field direct="true" name="mPropAddress" type="java.lang.String">
            <sql name="PROP_ADDR" type="varchar"/>
        </field>
        <field direct="true" name="mPropDate" type="java.util.Date">
            <sql name="PROP_LIST_DATE" type="date"/>
        </field>
        <field direct="true" name="mPropPrice" type="float">
            <sql name="PROP_ASK_PRICE" type="real"/>
        </field>
    </class>
</mapping>
```

For more information, see "Persistence Manager Configuration" (page 62).

Transaction Manager Configuration

WebObjects includes the Tyrex transaction manager. You configure it through the `TransactionManagerConfiguration.xml` file.

The only item you need to configure for the transaction manager is the domain. A transaction domain provides centralized management of transactions. It defines the policy for all transactions created from that domain, such as default timeout, maximum number of open transactions, support, and journaling. In addition, the domain maintains resource managers such as JDBC data sources and JCA (J2EE Connector Architecture) connectors.

This is an example of a `TransactionManagerConfiguration.xml` file:

```
<domain>
    <name>default</name>
    <resources>
        <dataSource>
            <name>DefaultDatabase</name>
            <jar>file:///tmp/oracle.jar</jar>
            <class>oracle.jdbc.xa.client.OracleXADataSource</class>
            <config>
                <serverName>HOSTNAME</serverName>
                <portNumber>PORTNUMBER</portNumber>
```

```
                <databaseName>DATABASENAME</databaseName>
                <driverType>thin</driverType>
                <user>ejb</user>
                <password>ejb</password>
            </config>
            <limits>
                <maximum>10</maximum>
                <minimum>0</minimum>
                <initial>0</initial>
                <maxRetain>300</maxRetain>
                <timeout>10</timeout>
            </limits>
        </dataSource>
    </resources>
</domain>
```

For details on how to write the transaction manager configuration file, see "Elements of the Transaction Manager Configuration File" (page 96).

Persistence Manager Configuration

Container-managed persistence is handled by the Castor JDO component. It generates SQL statements that the container uses to update your database. All you have to do is map your data store's table columns to entity beans' fields.

The persistence manager configuration files specify how the persistence manager obtains a connection to a data source, the mapping between Java classes and tables in the data source, and the service provider to use to talk to the data source.

These are supported database servers:

- Generic JDBC engine
- Oracle 7 and Oracle 8
- Sybase 11 and SQL Anywhere
- Microsoft SQL Server
- DB/2

- PostgreSQL 6.5 and 7
- Hypersonic SQL
- InstantDB
- Interbase
- MySQL
- SAP DB

You configure the persistence manager by editing three files:

- `CMPConfiguration.xml`

  This file defines the correspondence between table columns and the fields of your enterprise beans. It also defines how CMP beans are made persistent. This mapping is used in both local and global transaction configuration files.

- `LocalTransactionConfiguration.xml`

  This file defines the configuration that the persistence manager uses when a client uses an enterprise bean without a transaction context.

- `GlobalTransactionConfiguration.xml`

  This file defines the configuration that the persistence manager uses when a client uses an enterprise bean with a transaction context. This configuration requires that the data source be specified in the JNDI registry. The persistence manager creates the data source connection, which can be used in bean-managed as well as container-managed persistence beans.

Mapping Enterprise Beans to Data Stores

One of the tasks you need to perform to accomplish bean persistence is to map the enterprise bean fields to be persisted to table columns or other types of permanent storage. You accomplish this by editing the `CMPConfiguration.xml` file.

The Mapping File

The mapping information you enter in the `CMPConfiguration.xml` file is written from the point of view of the enterprise bean and describes how the contents of the bean's fields are translated to and from permanent storage.

This is an example of the contents of the `CMPConfiguration.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapping PUBLIC "-//EXOLAB/Castor Mapping DTD Version 1.0//EN"
                         "http://castor.exolab.org/mapping.dtd">
<mapping>
    <class key-generator="MAX" identity="mPropID"
name="webobjectsexamples.realestate.property.PropertyCMPBean">
        <map-to table="EJB_PROPERTY"/>
        <field direct="true" name="mPropID" type="java.lang.Integer">
            <sql name="PROP_ID" type="integer"/>
        </field>
        <field direct="true" name="mPropAddress" type="java.lang.String">
            <sql name="PROP_ADDR" type="varchar"/>
        </field>
        <field direct="true" name="mPropDate" type="java.util.Date">
            <sql name="PROP_LIST_DATE" type="date"/>
        </field>
        <field direct="true" name="mPropPrice" type="float">
            <sql name="PROP_ASK_PRICE" type="real"/>
        </field>
    </class>
</mapping>
```

For details in how to write the `CMPConfiguration.xml` file, see "Elements of the Component-Managed Persistence Configuration File" (page 78).

Primary Keys

The persistence manager can generate the values of identity properties automatically with the key generator. When the enterprise bean's `create` method is invoked, the persistence manager sets the value of the identity property to the value obtained from the key generator. The key generator can use one of several algorithms available to generate the value. You can use generic algorithms or

algorithms specific to your data source. For details on setting the algorithm to use for an enterprise bean's identity property, see "<class>" (page 81) and "<key-generator>" (page 86).

You can use the key generator only under the following conditions:

- The primary-key value is not determined from the arguments to the bean's `ejbCreate` method.
- The bean's identity can be determined through a single field of numeric (`byte` through `long`) or String type.

The following sections describe the key-generator algorithms you can use.

MAX

This generic algorithm fetches the maximum value of the primary key (`MAX`) and locks the record found until the end of the transaction. When the transaction ends, the value generated is `(MAX + 1)`. Because of the lock, concurrent transactions that use the same algorithm wait until the end of the original transaction to obtain a new primary-key value. Note that it is still possible to perform multiple inserts during the same transaction.

With this algorithm, duplicate-key exceptions are almost completely avoided. The only case in which they might occur is when inserting a row into an empty table because there are no rows to lock. In this case, the value generated is `1`.

This is an example of a definition of a key generator using the MAX algorithm:

```
<key-generator name="MAX">
    <param name="table" value="PERSON"/>
    <param name="key-column" value="PERSON_ID"/>
</key-generator>
```

HIGH/LOW

This generic algorithm needs an auxiliary table or sequence table containing a unique column (the key column) that stores table names and, a numeric (`integer`, `bigint`, or `numeric`) column used to reserve primary-key values.

The following table describes the parameters used by the HIGH/LOW key generator.

Table 5-2 HIGH/LOW key generator parameters

Parameter	Description	Use
`table`	Sequence-table name.	Mandatory
`key-column`	Name of the column containing table names.	Mandatory
`value-column`	Name of the column used to reserve primary-key values.	Mandatory
`grab-size`	Number of primary-key values the key generator reserves at a time.	Optional; default="`10`"
`same-connection`	Indicates whether the key generator must use the same connection when accessing the sequence table. Values: (`true` or `false`). Must be set to `true` when working in an EJB environment.	Optional; default="`false`"
`global`	Indicates whether the key generator produces globally unique keys. Values: (`true` or `false`).	Optional; default="`false`"

The first time the key generator is called, it finds the row for the target table in the sequence table, locks it, reads the last reserved primary-key value, increases it by the grab size (the number of primary-key values to reserve at a time), and unlocks the row. In subsequent requests for primary-key values for the same target table, the key generator provides primary-key values from the reserved values until it runs out. When it has no more primary-key values, it accesses the sequence table to obtain a new group of primary-key values.

Note: The sequence table must be in the same database as the table for which primary-key values are to be generated. When working with multiple databases, you must have one sequence table in each database that contains a table for which the key generator is to provide primary-key values.

If *grab-size* is set to `1`, the sequence tables contain the true maximum primary-key value at all times. In this case, the HIGH/LOW key generator is essentially equivalent to the MAX key generator.

If the *global* is set to `true`, the sequence table contains only one row instead of one row per table. The key generator uses this row for all tables.

UUID

This algorithm generates global unique primary-key values. The value generated is a combination of the host's IP address, the current time in milliseconds since 1970, and a static counter. The complete key consists of a 30-character, fixed-length string. This algorithm has no parameters. The primary-key column must be of type `char`, `varchar`, or `longvarchar`.

IDENTITY

The IDENTITY key generator can be used only with auto-increment primary-key columns (identities) in Sybase ASE/ASA, MS SQL Server, MySQL, and Hypersonic SQL.

After an insert, except when using MySQL or Hypersonic SQL, the key generator obtains the primary-key value from the `@@identity` system variable, which contains the last identity value for the current database connection. When using MySQL, the system function `LAST_INSERT_ID()` is used. For Hypersonic SQL, `IDENTITY()` is used.

SEQUENCE

This algorithm can be used with only Oracle, Oracle8i, PostgreSQL, Interbase, and SAP DB. It generates keys using sequences.

The following table describes the parameters for the SEQUENCE key generator.

Table 5-3

Parameter	Description	Use
`sequence`	Sequence name.	Optional; default="`{0}_seq`"
`returning`	RETURNING mode for Oracle8i. Values: (`true` or `false`)	Optional; default="`false`"
`increment`	Increment for Interbase.	Optional; default="`1`"
`trigger`	Indicates whether there is a trigger that generates primary-key values.	Optional; default="`false`"

Usually a sequence is used for only one table. Therefore, in general, you have to define one key generator per table. However, if you adhere to a naming convention for sequences, you can use one key generator for multiple tables.

For example, if you always obtain sequence names by adding `_seq` to the name of the corresponding table, you can set *sequence* to "`{0}_seq`" (the default).

The way this key generator performs its function depends on the data-source server being used.

With PostgreSQL, this key generator performs `SELECT nextval(sequence_name)` before the insert and produces the identity value that is then used when it performs `INSERT`.

With Interbase, the key generator performs `SELECT gen_id(sequence_name, increment) from rdb$database` before the insert.

With Oracle, by default (`returning="false"`) and with SAP DB, the key generator transforms the insert statement generated by the persistence manager to the form `INSERT INTO table_name (pk_name, ...) VALUES (sequence_name.nextval, ...)`, executes it, and then it performs `SELECT sequene_name.currval FROM table_name` to obtain the identity value.

With Oracle8i, when you set `returning` to "`true`", `RETURNING primary_key_name INTO ?` is appended to the insert statement shown above, which is a more efficient procedure to generate primary-key values. Therefore, the persistence manager fetches the identity value when it executes the insert statement (both the insertion and the procurement of the identity value occur in one statement).

If your table has an `on_Insert` trigger, like the one listed below, that already generates values for the table's primary key, you can set *trigger* to "`true`".

```
create or replace trigger "trigger_name"
before insert on "table_name" for each row
begin
    select "sequence_name".nextval into :new."pk_name" from dual;
end;
```

This prevents `"sequence_name".nextval` from being pulled twice: first during the insert and then in the trigger. It's also useful in combination with `returning="true"` for Oracle, in which case you may not specify the sequence name.

Defining Local and Global Data Stores

Local data-store configuration files, such as `LocalTransactionConfiguration.xml`, tell the transaction manager how to connect to a database. Global data-store configuration files tell the transaction manager how to locate a data store using JNDI. The transaction manager then uses the information in `TransactionManagerConfiguration.xml` to create database connections. The two files include the mapping between enterprise beans and tables in a data store.

The persistence manager can obtain a connection to a data store in one of three ways:

- using a JDBC 2.0 driver and URL
- using a JDBC 2.0 data source
- using a JNDI data source

If you are deploying the application inside a J2EE environment, you should use the JNDI method because it allows the application server to manage connection pooling and distributed transactions.

To allow for concurrent transactions and to ensure data integrity, two data-store definitions should never use overlapping mappings. This is an example of a local data store definition using an Oracle driver:

```
<database name="Local_TX_Database" engine="oracle">
    <driver class-name="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:thin:@<machine>:<port>:<database_name>">
        <param name="user" value="lucinda"
        <param name="password" value="tiger"/>
    </driver>
    <mapping href="Contents/Resources/CMPConfiguration.xml"/>
</database>
```

The following is the JNDI configuration of a global data store:

```
<database name="ebiz" engine="oracle">
    <jndi name="java:comp/env/jdbc/mydb"/>
    <mapping href="Contents/Resources/CMPConfiguration.xml"/>
</database>
```

For details on how to write the data-store configuration files, see "Elements of the Data-Store Configuration Files" (page 90).

Container Configuration

The `OpenEJBConfiguration.xml` file contains deployment information, as well as transaction and security details. Its contents are divided in two sections: containers, and facilities. WebObjects writes this file for you. However, you need to make additions, especially regarding the transaction type for methods and mapping physical roles to logical roles.

Containers Section

This section of the EJB configuration file holds four types of tags: `<containers>`, `<security-role>`, `<method-permission>`, and `<method-transaction>`.

The containers section (demarcated by the <containers> tag) can contain three types of tags: <stateless-session-container>, <stateful-session-container>, and <entity-container>. Each of these tags holds definitions for the corresponding types of enterprise beans: stateless session bean, stateful session bean, and entity bean (CMP and BMP).

One or more logical security roles are defined using <security-role> tags. Physical security roles are mapped to logical security roles in the facilities section of the file. You have to define the logical security roles that you want to use in your application. Then you assign those roles to the methods of the enterprise beans—using <method-permission> tags—as you see fit.

Note: WebObjects generates a default role and assigns it to all method permissions. You have to add the roles adequate for your situation and assign them to each of the methods of your enterprise beans through <method-permission> tags.

The <method-transaction> tag tells the container how to manage transactions for each method invocation. You must determine what kind of transaction attribute each enterprise bean's methods should have, and modify the contents of the <method-transaction> tag as appropriate. Table 5-4 provides a brief explanation of transaction attributes.

Table 5-4 Transaction attributes

Transaction attribute	Meaning
NotSupported	The current transaction is suspended until the method ends.
Supports	If in a transaction, the method is included in it.
Required	The method must be invoked within a transaction. Otherwise, a new transaction is created for the method.

Table 5-4 Transaction attributes

Transaction attribute	Meaning
`RequiresNew`	A new transaction is always created for the method.
`Mandatory`	The method must be invoked within a transaction. Otherwise, a `javax.transaction.TransactionRequiredException` is thrown.
`Never`	The method must never be invoked within a transaction. Otherwise, a `java.rmi.RemoteException` is thrown.

This is an example of the containers section of the EJB configuration file:

```
<container-system>
    <containers>
        <stateless-session-container>
            <container-name>Basic Stateless Container</container-name>
            <properties>
                <property>
                    <property-name>org/openejb/core/InstanceManager/
STRICT_POOLING</property-name>
                    <property-value>true</property-value>
                </property>
            </properties>
            <stateless-bean>
                <description>deployment descriptor for HelloBean</
description>
                <display-name>HelloBean</display-name>
                <ejb-deployment-id>HelloBean</ejb-deployment-id>
                <home>com.my.ejb.HelloHome</home>
                <remote>com.my.ejb.Hello</remote>
                <ejb-class>com.my.ejb.HelloBean</ejb-class>
                <transaction-type>Container</transaction-type>
            </stateless-bean>
        </stateless-session-container>
    </containers>
    <security-role>
        <role-name>everyone</role-name>
```

```
    </security-role>
    <method-permission>
        <role-name>everyone</role-name>
        <method>
            <ejb-deployment-id>HelloBean</ejb-deployment-id>
            <method-name>*</method-name>
        </method>
    </method-permission>
    <method-transaction>
        <method>
            <ejb-deployment-id>HelloBean</ejb-deployment-id>
            <method-intf>Remote</method-intf>
            <method-name>message</method-name>
            <method-params/>
        </method>
        <trans-attribute>NotSupported</trans-attribute>
    </method-transaction>
</container-system>
```

Facilities Section

This section of the EJB configuration file specifies the runtime environment: proxy-generation attributes, remote JNDI contexts, data-source connections, and J2EE services. The tags used are `<intra-vm-server>`, `<remote-jndi-contexts>`, `<connectors>`, and `<services>`, respectively. You should not edit this part of the `OpenEJBConfiguration.xml` file.

Using External Containers

You may want to use an external EJB container instead of an internal one in your bean-client applications when you already have a powerful, reliable container. In this case, you need to remove all the configuration files listed at the beginning of this chapter from your project.

To configure your application to use a single, external EJB container, you need to set system properties when you launch your application. You can set them through the command line. The following list details the properties you need to set for various EJB containers:

- OpenEJB

```
-Djava.naming.factory.initial=org.openorb.rmi.jndi.CtxFactory
-Djava.naming.provider.url=
corbaloc::1.2@$<HOST>:$<NAMESERVICE_PORT>/NameService"
-Dorg.omg.CORBA.ORBClass=org.openorb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=org.openorb.CORBA.ORBSingleton
-Djavax.rmi.CORBA.StubClass=org.openorb.rmi.system.StubDelegateImpl
-Djavax.rmi.CORBA.UtilClass=org.openorb.rmi.system.UtilDelegateImpl
-Djavax.rmi.CORBA.PortableRemoteObjectClass=
org.openorb.rmi.system.PortableRemoteObjectDelegateImpl
```

- iPlanet

```
-Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
-Djava.naming.provider.url=iiop://$<HOST>:$<NAMESERVICE_PORT>
```

- Web Logic

```
-Djava.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
-Djava.naming.provider.url=t3://$<HOST>:$<NAMESERVICE_PORT>
```

- WebSphere

```
-Djava.naming.factory.initial=
com.ibm.websphere.naming.WsnInitialContextFactory
-Djava.naming.provider.url=iiop://$<HOST>:$<NAMESERVICE_PORT>"
```

If you want to use more than one EJB container in your application, you'll have to set these properties through code. For example, to set the JNDI context for the Web Logic EJB container, you would add the following method:

```
// gets the JNDI context
public static Context initialContext() throws NamingException {
    Properties properties = new Properties();

    properties.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
```

```
    properties.put(Context.PROVIDER_URL, "t3://
$<HOST>:$<NAMESERVICE_PORT>");

    return new InitialContext(properties);
}
```

Configuration Reference

The elements (defined by tags) of an XML file can include attributes, other elements, or both. The sections below include tables that describe those elements. The attributes and sub-elements of a parent element can be tags or attributes: If the item's name is surrounded by < and >, the item is a sub-element (or tag); otherwise, it's an attribute. Table 6-1 describes the meaning of the symbols in the Use column in the tables that describe an element's members.

Table 6-1 Element usage symbols

Symbol in Use column	Meaning
Nothing	The tag or attribute is required by the parent tag.
?	The element or attribute can be omitted.
*	The element can be present zero or more times within the parent element.
+	The element must be present at least once within the parent element.

Elements of the Component-Managed Persistence Configuration File

The DTD for the `CMPConfiguration.xml` file is located at http://castor.exolab.org/mapping.dtd, and is shown in Listing 6-1.

Listing 6-1 DTD for CMPConfiguration.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mapping ( description?, include*, class*, key-generator* )>

<!ELEMENT include EMPTY>
<!ATTLIST include
    href  CDATA  #REQUIRED>

<!ELEMENT class ( description?, cache-type?, map-to?, field+ )>
<!ATTLIST class
    name            ID       #REQUIRED
    extends         IDREF    #IMPLIED
    depends         IDREF    #IMPLIED
    identity        CDATA    #IMPLIED
    access          ( read-only | shared | exclusive | db-locked )  "shared"
    key-generator   IDREF    #IMPLIED >

<!ELEMENT cache-type EMPTY>
<!ATTLIST cache-type
   type          ( none | count-limited | time-limited | unlimited ) "count-
limited"
    capacity       NMTOKEN  #IMPLIED>

<!ELEMENT map-to EMPTY>
<!ATTLIST map-to
    table      NMTOKEN  #IMPLIED
    xml        NMTOKEN  #IMPLIED
```

```
    ns-uri     NMTOKEN  #IMPLIED
    ns-prefix  NMTOKEN  #IMPLIED
    ldap-dn    NMTOKEN  #IMPLIED
    ldap-oc    NMTOKEN  #IMPLIED>

<!ELEMENT field ( description?, sql?, bind-xml?, ldap? )>
<!ATTLIST field
    name           NMTOKEN  #REQUIRED
    type           NMTOKEN  #IMPLIED
    required       ( true | false )  "false"
    direct         ( true | false )  "false"
    lazy           ( true | false )  "false"
    get-method     NMTOKEN  #IMPLIED
    set-method     NMTOKEN  #IMPLIED
    create-method  NMTOKEN  #IMPLIED
    collection     ( array | vector | hashtable | collection | set | map )
#IMPLIED>

<!ELEMENT sql EMPTY>
<!ATTLIST sql
    name        NMTOKEN  #IMPLIED
    type        CDATA    #IMPLIED
    many-key    NMTOKEN  #IMPLIED
    many-table  NMTOKEN  #IMPLIED
    dirty  ( check | ignore )  "check">

<!ELEMENT bind-xml EMPTY>
<!ATTLIST bind-xml
    name  NMTOKEN  #IMPLIED
    type  NMTOKEN  #IMPLIED
    matches  NMTOKEN  #IMPLIED
    node  ( attribute | element | text )  #IMPLIED>

<!ELEMENT ldap EMPTY>
<!ATTLIST ldap
    name  NMTOKEN  #IMPLIED>

<!ELEMENT key-generator  ( param* )>
<!ATTLIST key-generator
          name   CDATA  #REQUIRED
          alias  CDATA  #IMPLIED>
```

```
<!ELEMENT param EMPTY>
<!ATTLIST param
          name   CDATA  #REQUIRED
          value  CDATA  #REQUIRED>

<!ELEMENT description ( #PCDATA )>
```

The following sections describe the elements of the `CMPConfiguration.xml` file.

<bind-xml>

The attribute or element name and XML schema must be specified for all XML-dependent fields. The `node` attribute indicates whether the field maps to an attribute, another tag, or the textual content of this tag. Only simple types (primitives, date, string, and so on) can be used for attribute values. Only one field can be specified as the content model in a given object. Table 6-2 describes the `<bind-xml>` tag's members.

Table 6-2 Members of the <bind-xml> tag

Member	Use	Description
`name`	?	Table-column name.
`type`	?	
`matches`	?	
`node`	?	Value: `attribute`, `element`, or `text`.

<cache-type>

This tag tells the container how to cache instances of this enterprise bean. Table 6-3 describes the members of this element.

Table 6-3 Members of the <cache-type> tag

Member	Use	Description
`type`		Value: `none`, `count-limited`, `time-limited`, or `unlimited`. Default = `"count-limited"`.
`capacity`	?	The maximum number of instances of this bean the container is to create.

<class>

This tag describes the mapping between a Java class (enterprise bean implementation) and an SQL table, an XML element, an LDAP entry, or any other engine. To map a class into LDAP, an identity field must be specified.

A class is specified by its Java class name, including the package name; for example, `com.my.ejb.Person`. If a class extends another class for which a mapping file exists, you should use the `extends` attribute to include the class being extended. Do not use the `extends` attribute to describe Java class inheritance that is not reflected in any mapping.

The class mapping specifies each field in the class that is mapped to a table column. Fields that are not mapped are not stored, read, or otherwise processed. Table 6-4 describes the `<class>` tag's members.

Table 6-4 Members of the <class> tag

Member	Use	Description
`name`		Class name.
`extends`	?	Implied by the persistence manager. It's the name of the class this class extends. Used only if this class extends another class for which mapping information is provided.
`depends`	?	Implied by the persistence manager.
`identity`	?	Implied by the persistence manager.
`access`		Value: `read-only`, `shared`, `exclusive` or `db-locked`. Default = `"shared"`
`key-generator`	?	Name or alias of the key generator to use. Use only for classes with single-property, numeric ID fields. If your class uses a compound primary key or the primary key contains strings, you must use a custom key generator; that is, the bean itself must create the primary-key values. See "<key-generator>" (page 86).
`<description>`	?	Optional class description.
`<cache-type>`	?	See "<cache-type>" (page 81).
`<map-to>`	?	Used if the name of the element this class maps to is not the same as the name of the class. By default, the persistence manager infers the name of the element from the name of the class: a class named `SocialEvent` is mapped to an element called `social-event`. See "<map-to>" (page 88).
`<field>`	+	Describes the properties of an enterprise bean. See "<field>" (page 83).

<field>

This tag specifies the mapping between an enterprise bean's field and an SQL table column, an XML element or attribute, an LDAP attribute, and so on. Table 6-5 describes its members.

Table 6-5 Members of the <field> tag

Item	Use	Description
`name`		Name of the enterprise bean's field being mapped.
`type`	?	Java type of the field. For example, `java.lang.Integer`.
`required`	?	Value: `true` or `false`. Default = "`false`". Indicates whether the field is optional or required.
`direct`	?	Value: `true` or `false`. Default = "`false`".
`lazy`	?	Value: `true` or `false`. Default = "`false`".
`get-method`	?	Implied by the persistence manager.
`set-method`	?	Implied by the persistence manager.
`create-method`	?	Implied by the persistence manager.
`collection`	?	Value: `array`, `vector`, `hashtable`, `collection`, `set`, or `map`. Implied by the persistence manager.
`<description>`	?	Optional field description.
`<sql>`	?	See "<sql>" (page 89).
`<bind-xml>`	?	See "<bind-xml>" (page 80).
`<ldap>`	?	See "<ldap>" (page 87).

The mapping is specified from the perspective of the bean's implementation class. The field name is required even if no such field exists in the class in order to support field references. A field is an abstraction of an enterprise bean's property: It can refer to a property directly (by mapping to a `public` instance variable, not `static` nor `transient`) or indirectly by using accessor methods.

Unless specified otherwise, the persistence manager accesses the field through *get* and *set* methods, whose names are derived from the field name. For example, for a field called `lastName`, the accessors `String getName()` and `void setName(String)` are used. Collection fields require only a *get* method, except an array requires both a *get* and a *set* method. If the accessors are specified through the `get-method` and `set-method` attributes, the persistence manager accesses the field only through those methods. The methods must be `public` and not `static`.

If the `direct` attribute is `true`, the field is accessed directly. The field must be `public`, not `static` nor `transient`.

The `type` attribute indicates the type of the instance variable being mapped or the type of each a collection's elements. You can use fully qualified class names or a short name, as Table 6-6 illustrates.

Table 6-6 Values for the type attribute of the <field> tag for CMP beans

Short name	Fully qualified name
`other`	`java.lang.Object`
`string`	`java.lang.String`
`integer`	`integer`
`long`	`long`
`boolean`	`boolean`
`double`	`double`
`float`	`float`
`big-decimal`	`java.math.BigDecimal`
`byte`	`byte`
`date`	`java.util.Date`
`short`	`short`
`char`	`char`
`bytes`	`byte[]`

Table 6-6 Values for the type attribute of the <field> tag for CMP beans

Short name	Fully qualified name
chars	char[]
strings	string[]
locale	java.lang.Locale

If the field is a collection, you specify the collection type through the `collection` attribute and the type of each element of the collection through the `type` attribute. Use the following table to determine the appropriate value for the `collection` attribute.

Table 6-7 Values for the collection attribute of the <field> tag in CMP beans

Collection attribute value	Type of collection	Default implementation
array	<type>[]	<type>[]
vector	java.util.Vector	java.util.Vector
hashtable	java.util.Hashtable	java.util.Hashtable
collection	java.util.Collection	java.util.Arraylist
set	java.util.Set	java.util.Hashset
map	java.util.Map	java.util.Hashmap

The "Default implementation" column indicates the type used if the object holding the collection is `null` and needs to be instantiated. For `hashtable` and `map` collections, the persistence manager adds an object with the `put(Object, Object)` method: The object added is both the key and the value.

Table 6-5 (page 83) describes the members of the `<field>` tag.

<key-generator>

This tag specifies parameters for the key generator (if needed). For example, to obtain sequential values from the table SEQTAB, use

```
<key-generator name="SEQUENCE">
    <param name="table" value="SEQTAB">
    <param name="global" value="0">
</key-generator>
<class key-generator="SEQUENCE">
    ...
</class>
```

If you have to use several key generators of the same type for the same data store, use aliases:

```
<key-generator name="SEQUENCE" alias="seq1">
    <param name="table" value="SEQTAB">
    <param name="global" value="0">
</key-generator>
<key-generator name="SEQUENCE" alias="seq2">
    <param name="table" value="SEQGLOBAL">
    <param name="global" value="1">
</key-generator>
<class key-generator="seq2">
    ...
</class>
```

Table 6-8 describes the members of the <key-generator> tag.

Table 6-8 Members of the <key-generator> tag

Member	Use	Description
name		Sequence-table name.
alias	?	Additional identifier for the key generator.
<param>	*	See "<param>" (page 89).

Table 6-9 lists the key-generator names supported in the persistence manager.

Table 6-9 Key-generator names supported in the persistence manager

Name	Description
`MAX`	*MAX(pk) + 1* generic algorithm.
`HIGH/LOW`	HIGH/LOW generic algorithm.
`UUID`	UUID generic algorithm.
`IDENTITY`	Supports auto-increase identity fields in Sybase ASE/ASA, MS SQL Server, MySQL, and Hypersonic SQL.
`SEQUENCE`	Supports the SEQUENCE algorithm in Oracle, PostgreSQL, Interbase, and SAP DB.

<ldap>

This tag contains field mapping information for fields mapped to LDAP resources. Table 6-10 describes its members.

Table 6-10 Members of the <ldap> tag

Member	Use	Description
`name`	?	LDAP-resource name.

<map-to>

This tag specifies the mapping between an enterprise bean and an SQL table. Table 6-11 describes the tag's members.

Table 6-11 Members of the <map-to> tag

Item	Use	Description
`table`	?	SQL table name.
`xml`	?	
`ns-uri`	?	
`ns-prefix`	?	
`ldap-dn`	?	
`ldap-oc`	?	

<mapping>

The `<mapping>` tag is the root element of the entire file. It defines a collection of class mappings. Its members are described in Table 6-12.

Table 6-12 Members of the <mapping> tag

Member	Use	Description
`<description>`	?	Optional description of the mapping.
`<include>`	*	Used to include other mappings in this mapping. The tag's sole member is the `href` attribute, set to the URL that indicates the location of the mapping file.
`<class>`	*	See "<class>" (page 81).
`<key-generator>`	*	See "<key-generator>" (page 86).

<param>

This tag is used to provide named parameters to the containing element. Table 6-13 describes the members of this tag.

Table 6-13 Members of the <param> tag

Member	Use	Description
`name`		Parameter name.
`value`		Parameter value.

<sql>

This tag provides field mapping information that is relevant only for fields mapped to SQL tables. The type can be the proper Java-class type returned by the JDBC driver or the SQL type without precision, for example, "`java.math.BigDecimal`" or "`numeric`". However, the type could contain the parameter for the SQL-to-Java type convertors in square brackets, for example, "`char[01]`" for `false=0`, `true=1` conversion from the `boolean` Java type to the `char` SQL type.

Table 6-14 Members of the <sql> tag

Member	Use	Description
`name`	?	Table-column name.
`type`	?	SQL type of the column.
`many-key`	?	
`many-table`	?	
`dirty`	?	Value: `check` or `ignore`. Default = "`check`".

Elements of the Data-Store Configuration Files

The DTD for the local and global transaction configuration files can be found at http://castor.exolab.org/jdo-conf.dtd, and is shown in Listing 6-2.

Listing 6-2 DTD for LocalTransactionConfiguration.xml and GlobalTransactionConfiguration.xml

```
<!ELEMENT database ( ( driver | data-source | jndi )?, mapping+ )>
<!ATTLIST database
          name ID      #REQUIRED
          engine CDATA "generic">

<!ELEMENT mapping EMPTY>
<!ATTLIST mapping
          href CDATA #REQUIRED>

<!ELEMENT driver ( param* )>
<!ATTLIST driver
          url        CDATA #REQUIRED
          class-name CDATA #IMPLIED>

<!ELEMENT param EMPTY>
<!ATTLIST param
          name  CDATA #REQUIRED
          value CDATA #REQUIRED>

<!ELEMENT data-source ( params )>
<!ATTLIST data-source
          class-name CDATA #REQUIRED>


<!ELEMENT jndi ANY>
<!ATTLIST jndi
          name CDATA #REQUIRED>
```

The following sections describe the elements of the `LocalTransactionConfiguration.xml` and `GlobalTransactionConfiguration.xml` files.

<data-source>

This tag specifies the JDBC 2.0 data-source used to obtain new connections to the database server. Table 6-15 describes its members.

Table 6-15 Members of the <data-source> tag

Member	Use	Description
`class-name`		The class name of the data.
`<params>`		Depends on the database server.

This is an example of a data-source definition:

```
<data-source class-name="org.postgresql.PostgresqlDataSource">
    <param host="host" database="db"
           user="user" password="secret"/>
</data-source>
```

<database>

This tag specifies the database server that the persistence manager uses to establish connections to a database. Table 6-16 describes its members.

Table 6-16 Members of the <database> tag

Item	Use	Description
`name`		Database name. Used by the application to connect to a database.
`engine`	*	Specifies the persistence engine (service provider) for this database server. Default = "`generic`". See "engine" (page 94).
`<driver>` or `<data-source>` or `<jndi>`	?	Determines the type of connection to use to link to the datasource. Only one can be used. See "<driver>" (page 93), "<data-source>" (page 91), and "<jndi>" (page 95). You can see example definitions for common databases below.
`<mapping>`	+	Used to include the mappings to use with this database. The element's sole member is the `href` attribute, set to the URL that indicates the location of the mapping file.

These are the connection definitions for common database servers:

- Sybase jConnect

```
<data-source classname="com.sybase.jdbc2.jdbc.SybDataSource">
    <params user="user" password="secret"
            port-number="4100" server-name="host"/>
</data-source>
```

- Oracle thin driver

```
<driver class-name="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@host:post:SID?user=shelley">
    <param name="password" value="tiger"/>
</driver>
```

- PostgreSQL

```
<data-source class-name="org.postgresql.PostgresqlDataSource">
    <params host="host" database="db"
            user="user" password="secret"/>
</data-source>
```

- InstantDB

```
<driver class-name="org.enhydra.instantdb.jdbc.idbDriver"
    url="jdbc:idb:C:\\castor-0.8.8\\db\\test\\test.prp">
    <param name="user" value=""/>
    <param name="password" value=""/>
</driver>
```

<driver>

This tag specifies the JDBC 2.0 driver used to obtain new connections to the data-source server. Table 6-17 describes the members of this element.

Table 6-17 Members of the <driver> tag

Member	Use	Description
`class-name`	?	The class name of the driver; it must be located in the `classpath` environment variable. Can be omitted if it's specified by other means, such as by a properties file, `Class.forName()`, and so on.
`url`		Locates the driver and provides access properties.
`<param>`	*	Additional properties required by the driver. See "<param>" (page 89).

engine

This attribute specifies a database engine. Table 6-18 lists the engines supported by the persistence manager.

Table 6-18 Database engines supported by the persistence manager

Engine name	Database
`generic`	Generic JDBC support
`oracle`	Oracle 7 and Oracle 8
`sybase`	Sybase 11
`sql-server`	Microsoft SQL Server
`db2`	DB/2
`informix`	Informix
`postgresql`	PostgreSQL 7.1
`hsql`	Hypersonic SQL
`instantdb`	InstantDB
`interbase`	Interbase
`mysql`	MySQL
`sapdb`	SAP DB

<jndi>

This tag specifies the database you want to connect to. The persistence manager uses the JNDI environment naming context (ENC) to obtain a connection to the database. Table 6-19 describes this element's only attribute.

Table 6-19 Members of the <jndi> tag

Member	Use	Description
`name`		JNDI name of the database.

This is an example of a JNDI datasource specification:

```
<jndi name="java:comp/env/jdbc/mydb"/>
```

Elements of the Transaction Manager Configuration File

The following sections describe the elements of the `TransactionManagerConfiguration.xml` file.

<config>

The `<config>` tag provides the configuration for a JDBC data source. Table 6-20 describes its members.

Table 6-20 Data members of the <config> tag

Member	Use	Description
`<serverName>`		Server name.
`<portNumber>`		Port number.
`<databaseName>`	?	Database name.
`<driverType>`	?	Driver type. Value: `thin`.
`<user>`		User ID.
`<password>`		Password.

<connector>

The <connector> tag specifies a database-connection factory. Table 6-21 describes its members.

Table 6-21 Members of the <connector> tag

Member	Use	Description
<name>		Connector name.
<jar>		Connector JAR filename.
<paths>	?	Paths to additional JAR and dependent files.
<config>	?	See "<config>" (page 96).
<limits>	?	See "<limits>" (page 98).

<dataSource>

The <dataSource> tag contains a specification for a JDBC data source. Table 6-22 describes the members of this element.

Table 6-22 Members of the <dataSource> tag

Member	Use	Description
<name>		Data-source name.
<jar>		Data-source JAR filename.
<paths>	?	Paths to additional JAR and dependent files.
<class>		Class name of the data-source implementation.
<config>	?	See "<config>" (page 96).
<limits>	?	See "<limits>" (page 98).

<domain>

The <domain> tag is the root tag of the entire file. Table 6-23 describes the members of this element.

Table 6-23 Members of the <domain> tag

Member	Use	Description
<name>		Domain name.
<maximum>	?	Maximum number of open transactions allowed.
<timeout>	?	Default timeout (in seconds) for transactions.
<journalFactory>	?	Transaction journal factory's implementation class.
<resources>	?	See "<resources>" (page 99).

<limits>

The <limits> tag provides resource limits for a data source or a connector. Table 6-24 describes its members.

Table 6-24 Data members of the <limits> tag

Member	Use	Description
<maximum>	?	Maximum number of connections allowed.
<minimum>	?	Minimum number of connections allowed.
<initial>	?	Initial pool size.
<maxRetain>	?	Maximum period (in seconds) to retain open connections.
<timeout>	?	Maximum timeout (in seconds) to wait for a new connection.
<trace>	?	Turns tracing on ("true") or off ("false").

<resources>

The <resources> tag is the top-level tag of a list of JDBC data sources and JCA connectors. Table 6-25 describes the members of the <resources> tag.

Table 6-25 Members of the <resources> tag

Member	Use	Description
<dataSource>	*	See "<dataSource>" (page 97).
<connector>	*	See "<connector>" (page 97).

Elements of the Container Configuration File

The DTD for the deployment configuration file, OpenEJBConfiguration.xml, is stored in /System/Library/WebObjects/JavaApplications/OpenEJBTool.woa/Contents/Resources. The DTD is also added to the Resources group of an enterprise-bean–client application project. The file, called openejb_config.dtd, is shown in Listing 6-3. (You must never edit this file.)

Listing 6-3 DTD for OpenEJBConfiguration.xml

```
<?xml encoding="US-ASCII"?>
<!ELEMENT class-name (#PCDATA)>
<!ELEMENT cmp-field-name (#PCDATA)>
<!ELEMENT codebase (#PCDATA)>
<!ELEMENT connection-manager (connection-manager-id, class-
name,properties?)>
<!ELEMENT connection-manager-id (#PCDATA)>
<!ELEMENT connector (connector-id, connection-manager-id, managed-connection-
factory)>
<!ELEMENT connector-id (#PCDATA)>
```

```
<!ELEMENT connectors (connector*, connection-manager+)>
<!ELEMENT container-name (#PCDATA)>
<!ELEMENT container-system (containers, security-role+, method-permission+,
method-transaction+)>
<!ELEMENT containers  (stateful-session-container|stateless-session-
container|entity-container)+>
<!ELEMENT description (#PCDATA)>
<!ELEMENT display-name (#PCDATA)>
<!ELEMENT ejb-class (#PCDATA)>
<!ELEMENT ejb-deployment-id (#PCDATA)>
<!ELEMENT ejb-ref (ejb-ref-name, home, ejb-ref-location)>
<!ELEMENT ejb-ref-location (ejb-deployment-id | (remote-ref-name, jndi-
context-id))>
<!ELEMENT ejb-ref-name (#PCDATA)>
<!ELEMENT entity-bean (description?, display-name?, small-icon?,large-icon?,
ejb-deployment-id, home, remote, ejb-class, persistence-type, prim-key-class,
reentrant, cmp-field-name*, primkey-field?, jndi-enc?, security-role-ref*,
query*)>
<!ELEMENT entity-container (codebase?, description?, display-name?,
container-name, properties?, entity-bean+)>
<!ELEMENT env-entry (env-entry-name, env-entry-type, env-entry-value)>
<!ELEMENT env-entry-name (#PCDATA)>
<!ELEMENT env-entry-type (#PCDATA)>
<!ELEMENT env-entry-value (#PCDATA)>
<!ELEMENT facilities (intra-vm-server, remote-jndi-contexts?, connectors?,
services)>
<!ELEMENT factory-class (#PCDATA)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT intra-vm-server (proxy-factory, codebase?, properties?)>
<!ELEMENT jndi-context (jndi-context-id, properties)>
<!ELEMENT jndi-context-id (#PCDATA)>
<!ELEMENT jndi-enc (env-entry*, ejb-ref*, resource-ref*)>
<!ELEMENT large-icon (#PCDATA)>
<!ELEMENT logical-role-name (#PCDATA)>
<!ELEMENT managed-connection-factory (class-name, properties?)>
<!ELEMENT method (description?, ejb-deployment-id?, method-intf?, method-
name, method-params?)>
<!ELEMENT method-intf (#PCDATA)>
<!ELEMENT method-name (#PCDATA)>
<!ELEMENT method-param (#PCDATA)>
<!ELEMENT method-params (method-param*)>
```

```
<!ELEMENT method-permission (description?, role-name+, method+)>
<!ELEMENT method-transaction (description?, method+, trans-attribute)>
<!ELEMENT openejb (container-system, facilities)>
<!ELEMENT persistence-type (#PCDATA) >
<!ELEMENT physical-role-name (#PCDATA)>
<!ELEMENT prim-key-class (#PCDATA)>
<!ELEMENT primkey-field (#PCDATA)>
<!ELEMENT properties (property+)>
<!ELEMENT property (property-name, property-value)>
<!ELEMENT property-name (#PCDATA)>
<!ELEMENT property-value (#PCDATA)>
<!ELEMENT proxy-factory (#PCDATA)>
<!ELEMENT query (description?, method, query-statement)>
<!ELEMENT query-statement (#PCDATA)>
<!ELEMENT reentrant (#PCDATA)>
<!ELEMENT remote (#PCDATA)>
<!ELEMENT remote-jndi-contexts (jndi-context+)>
<!ELEMENT remote-ref-name (#PCDATA)>
<!ELEMENT res-auth (#PCDATA)>
<!ELEMENT res-id (#PCDATA)>
<!ELEMENT res-ref-name (#PCDATA)>
<!ELEMENT res-type (#PCDATA)>
<!ELEMENT resource (description?, res-id, properties)>
<!ELEMENT resource-ref (description?, res-ref-name, res-type, res-auth, (res-
id | properties | connector-id))>
<!ELEMENT role-link (#PCDATA)>
<!ELEMENT role-mapping (logical-role-name+, physical-role-name+)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT security-role (description?, role-name)>
<!ELEMENT security-role-ref (description?, role-name, role-link)>
<!ELEMENT security-service (description?, display-name?, service-name,
factory-class, codebase?,properties?, role-mapping+)>
<!ELEMENT security-service-name (#PCDATA)>
<!ELEMENT service-name (#PCDATA)>
<!ELEMENT services (security-service, transaction-service)>
<!ELEMENT small-icon (#PCDATA)>
<!ELEMENT stateful-bean (description?, display-name?, small-icon?,large-
icon?, ejb-deployment-id, home, remote, ejb-class, transaction-type, jndi-
enc?, security-role-ref*)>
<!ELEMENT stateful-session-container (codebase?, description?, display-name?,
container-name, properties?, stateful-bean+)>
```

```
<!ELEMENT stateless-bean (description?, display-name?, small-icon?,large-
icon?, ejb-deployment-id, home, remote, ejb-class, transaction-type, jndi-
enc?, security-role-ref*)>
<!ELEMENT stateless-session-container (codebase?, description?, display-
name?, container-name, properties?, stateless-bean+)>
<!ELEMENT trans-attribute (#PCDATA)>
<!ELEMENT transaction-service (description?, display-name?, service-name,
factory-class, codebase?, properties?) >
<!ELEMENT transaction-service-name (#PCDATA)>
<!ELEMENT transaction-type (#PCDATA)>
```

The following sections describe the elements of the `OpenEJBConfiguration.xml` file.

<connection-manager>

This tag specifies a connection manager. Table 6-26 describes its members.

Table 6-26 Members of the <connection-manager> tag

Member	Use	Description
`<connection-manager-id>`		Name of the connection manager.
`<class-name>`		Class name of the data source.
`<properties>`		Properties required by the data source.

<connector>

This tag defines a connector. Table 6-27 describes its members.

Table 6-27 Members of the <connector> tag

Member	Use	Description
`<connector-id>`		Name of the connector.
`<connection-manager-id>`		Specifies a connection manager. See "<connection-manager>" (page 102).
`<managed-connection-factory>`	*	See "<managed-connection-factory>" (page 110).

<connectors>

This tag encloses connectors or connection managers. Table 6-28 describes its members.

Table 6-28 Members of the <connectors> element

Member	Use	Description
`<connector>`	*	See "<connector>" (page 103).
`<connection-manager>`	+	See "<connection-manager>" (page 102).

<container-system>

This tag delimits the container configuration section of the deployment configuration file. Table 6-29 describes its members.

Table 6-29 Members of the <container-system> tag

Member	Use	Description
`<containers>`		See "<dataSource>" (page 97).
`<security-role>`	+	See "<connector>" (page 97).
`<method-permission>`	+	Assigns a logical role to methods of the enterprise beans defined in the `containers` element.
`<method-transaction>`	+	Specifies a method's transaction attribute.

<containers>

This tag encloses containers for the three types of enterprise beans: stateless session beans, stateful session beans, and entity beans. Table 6-30 describes its members.

Table 6-30 Members of the <containers> tag

Member	Use	Description

`<stateful-session-container>`
`<stateless-session-container>`
`<entity-container>`
	+	At least one of these items must be present.

<ejb-ref>

This tag defines a reference to a bean so that the bean can be accessed using JNDI calls. Table 6-31 describes its members.

Table 6-31 Members of the <ejb-ref> tag

Member	Use	Description
`<ejb-ref-name>`		JNDI name for the bean. For example, `ejb/agent/Agent`.
`<home>`		Home interface of the bean. For example, `webobjectsexamples.realestate.agent.AgentHome`.
`<ejb-ref-location>`		See "<ejb-ref-location>" (page 105).

<ejb-ref-location>

This tag identifies a bean through its name (using its `<ejb-deployment-id>` member) or through its remote interface and JNDI context ID. Table 6-32 describes its members.

Table 6-32 Members of the <ejb-ref-location> tag

Member	Use	Description
`<ejb-deployment-id>` or (`<remote-ref-name>`, `<jndi-context-id>`)		Either `<ejb-deployment-id>` or `<remote-ref-name>` and `<jndi-context-id>` must be specified.

<entity-bean>

This tag defines an entity session bean. Table 6-33 describes its members.

Table 6-33 Members of the <entity-bean> tag

Member	Use	Description
`<description>`	?	Description for the bean.
`<display-name>`	?	
`<small-icon>`	?	
`<large-icon>`	?	
`<ejb-deployment-id>`		Name of the bean.
`<home>`		Home interface (for example, `com.my.ejb.PersonHome`).
`<remote>`		Remote interface (for example, `com.my.ejb.Person`).
`<ejb-class>`		Implementation class (for example, `com.my.ejb.PersonBean`).
`<persistence-type>`		Value: `Container` or `Bean`.
`<prim-key-class>`		Fully qualified class name of the primary key.
`<reentrant>`		Value: `true` or `false`. Should be `false`.
`<cmp-field-name>`	*	Container-managed–persistence field name.
`<primkey-field>`	?	Primary-key field name.
`<jndi-enc>`	?	See "<jndi-enc>" (page 109).
`<security-role-ref>`	*	See "<security-role-ref>" (page 118).
`<query>`	*	Specifies a query for a finder method.

<entity-container>

This tag defines an entity-bean container and encloses the definition of entity beans. Table 6-34 describes its members.

Table 6-34 Members of the <entity-container> tag

Member	Use	Description
<codebase>	?	
<description>	?	Description of the container.
<display-name>	?	
<container-name>		Name for the container.
<properties>	?	Used to tell the container how to handle instances of entity beans. See "<properties>" (page 113).
<entity-bean>	+	Entity bean definitions. See "<entity-bean>" (page 106).

<env-entry>

This tag defines an environment variable and its value (which can be accessed by other beans through JNDI). Table 6-35 describes its members.

Table 6-35 Members of the <env-entry> tag

Member	Use	Description
<env-entry-name>		Name of the variable.
<env-entry-type>		Java type of the variable.
<env-entry-value>		Value for the variable.

<facilities>

This tag specifies the runtime environment: proxy-generation attributes, remote JNDI contexts, data-source connections, and J2EE services. You should not change the information within `<facilities>` and `</facilities>` tags. Table 6-36 describes its members.

Table 6-36 Members of the <facilities> tag

Member	Use	Description
`<intra-vm-server>`		
`<remote-jndi-contexts>`	?	
`<connectors>`	?	
`<services>`		

<jndi-context>

This tag defines one external JNDI context to be used by the application. Table 6-37 describes its members.

Table 6-37 Members of the <jndi-context> tag

Member	Use	Description
`<jndi-context-id>`		Name of the JNDI context.
`<properties>`		Required properties.

<jndi-enc>

This tag encloses naming information so that this bean can be located through JNDI. Table 6-38 describes the members of the <jndi-enc> tag.

Table 6-38 Members of the <jndi-enc> tag

Member	Use	Description
<env-entry>	*	
<ejb-ref>	*	Defines a reference to this bean. See "<ejb-ref>" (page 105).
<resource-ref>	*	Defines the beans data source. See "<resource-ref>" (page 115).

<intra-vm-server>

This tag specifies the dynamic factory proxy to use to create client proxies of the real EJB objects. Table 6-39 describes its member.

Table 6-39 Member of the <intra-vm-server> tag

Member	Use	Description
<proxy-factory>		Dynamic proxy factory. Values: org.openejb.util.proxy.jdk13.Jdk13ProxyFactory or org.openejb.util.proxy.DynamicProxy.

<managed-connection-factory>

This tag defines a managed-connection factory. Table 6-43 describes its members.

Table 6-40 Members of the <managed-connection-factory> tag

Member	Use	Description
`<class-name>`		Class name of the data source.
`<properties>`	?	Properties required by the data source.

<method>

This tag specifies a home or remote interface method of an enterprise bean. Table 6-41 describes its members.

Table 6-41 Members of the <method> tag

Member	Use	Description
`<description>`	?	Description for the method.
`<ejb-deployment-id>`	?	Must specify the ID (name) of one of the enterprise beans declared in the `<container-system>` tag. If this element isn't specified, this method declaration applies to all matching bean methods (home and remote interfaces) of all the enterprise beans defined in the `<container-system>` tag.
`<method-intf>`	?	Value: `Home` or `Remote`. Distinguishes between a method with the same signature that is defined in both the home and remote interface.
`<method-name>`		Specifies the method name.
`<method-params>`	?	Identifies a single method among multiple methods with an overloaded method name. If the method takes no input arguments, this element can be empty or omitted.

These examples of the three possible styles of the <method> tag's syntax:

- Referring to all the methods (home and remote interfaces) defined within the <container-system> tag.

```
<method>
    <method-name>*</method-name>
</method>
```

- Referring to a specific method defined within the <container-system> tag.

```
<method>
    <method-name>METHOD</method-name>
</method>
```

- Referring to a single method within a set of methods (home and remote interfaces) with an overloaded name.

```
<method>
    <method-name>METHOD</method-name>
    <method-params>
        <method-param>PARAM-1</method-param>
        <method-param>PARAM-2</method-param>
                ...
        <method-param>PARAM-n</method-param>
    </method-params>
</method>
```

<method-params>

This tag is used when further identification of a method is needed due to method-name overloading. Table 6-42 describes its members.

Table 6-42 Members of the <method-params> tag

Member	Use	Description
<method-param>	*	Fully qualified Java type. Specify arrays by following the array element's type with one or more pairs of square brackets (for example, int[]).

<method-permission>

This tag maps security roles to methods. Table 6-43 describes its members.

Table 6-43 Members of the <method-permission> tag

Member	Use	Description
`<description>`	?	Description for the method permission.
`<role-name>`	+	Logical role name corresponding to a `<security-role>` tag.
`<method>`	+	See "<method>" (page 110).

<method-transaction>

This tag specifies how the container manages transaction scopes when delegating a method invocation to an enterprise bean's implementation class. Table 6-44 describes its members.

Table 6-44 Members of the <method-transaction> tag

Member	Use	Description
`<description>`	?	Description for the method and the transaction.
`<method>`	+	Methods to apply the transaction type to.
`<trans-attribute>`		Value: `NotSupported`, `Supports`, `Required`, `RequiresNew`, `Mandatory`, `Never`, or `Bean`.

<openejb>

This is the root tag of the deployment configuration file. Table 6-45 describes its members.

Table 6-45 Members of the <openejb> tag

Member	Use	Description
`<container-system>`		See "<container-system>" (page 104).
`<facilities>`		See "<facilities>" (page 108).

<properties>

This tag encloses a set of property-value definitions. Table 6-46 describes its member.

Table 6-46 Member of the <properties> tag

Member	Use	Description
`<property>`		See "<property>" (page 113).

<property>

This tag encloses a property-value definition. Table 6-47 describes its members.

Table 6-47 Members of the <property> tag

Member	Use	Description
`<property-name>`		The name of the property.
`<property-value>`		The value for the property.

<query>

This tag can be used to declare a query statement and bind it to a specific finder method. The value can be retrieved using the `org.openejb.core.DeploymentInfo.getQuery` method. Table 6-48 describes the members of the `<query>` tag.

Table 6-48 Members of the <query> tag

Member	Use	Description
`<description>`	?	Description for the query.
`<method>`		The `<ejb-deployment-id>` tag of `<method>` is ignored (should not be used). See "<method>" (page 110).
`<query-statement>`	*	SQL statement.

<remote-jndi-contexts>

This tag groups external JNDI contexts. Table 6-49 describes its members.

Table 6-49 Member of the <remote-jndi-contexts> tag

Member	Use	Description
`<jndi-context>`	+	See "<jndi-context>" (page 108).

<resource>

This tag defines a resource. Table 6-50 describes its members.

Table 6-50 Member of the <resource> tag

Member	Use	Description
`<description>`	?	Description for the resource.
`<res-id>`		Maps this resource to a `<connector-id>` element in the corresponding `<connectors>` section.
`<properties>`		See "<properties>" (page 113).

<resource-ref>

This tag specifies a reference to an external resource. Table 6-51 describes its members.

Table 6-51 Members of the <resource-ref> tag

Member	Use	Description
`<description>`	?	Description for the resource.
`<res-ref-name>`		Specifies the name of a resource manager connection-factory reference (for example, `comp/env/jdbc/Employee`).

Table 6-51 Members of the <resource-ref> tag

Member	Use	Description
<res-type>		Specifies the type of the data source, that is, the Java class or interface expected to be implemented by the data source (for example, javax.sql.DataSource).
<res-auth>		Value: Application or Container. Specifies who signs on to the resource manager: the enterprise bean or the container.
<res-id> or <connector-id> or <properties>		You can map this resource reference to a resource, a connector, or to a set of properties.

This is an example of a <resource-ref> definition using properties:

```
<resource-ref>
    <res-ref-name>comp/env/jdbc/Employee</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <properties>
        <property>
            <property-name>url</property-name>
            <property-value>jdbc:odbc:orders</property-value>
        </property>
        <property>
            <property-name>username</property-name>
            <property-value>Admin</property-value>
        </property>
        <property>
            <property-name>password</property-name>
            <property-value></property-value>
        </property>
    </properties>
</resource-ref>
```

<role-mapping>

This tag maps a logical security role to a physical security role. Table 6-52 describes its members.

Table 6-52 Members of the <role-mapping> tag

Member	Use	Description
`<logical-role-name>`	+	Logical security-role name.
`<physical-role-name>`	+	Physical security-role name.

<security-role>

This tag defines a logical role name. Table 6-53 describes its members.

Table 6-53 Members of the <security-role> tag

Member	Use	Description
`<description>`	?	Description of the logical role.
`<role-name>`		Logical role name (for example, `everyone` or `admin`).

<security-role-ref>

This tag specifies a security-role reference. Table 6-54 describes its members

Table 6-54 Members of the <security-role-ref> tag

Member	Use	Description
`<description>`	?	Description of the security role.
`<role-name>`		Security-role name used in code. It must be the String used as the argument in the invocation of the `isCallerInRole(String)` method of EJBContext.
`<role-link>`		Name of a security role (`<security-role>` tag). Links this security-role reference to a defined security role. See "<security-role>" (page 117).

<security-service>

This tag defines a security service. Table 6-55 describes its members.

Table 6-55 Members of the <security-service> tag

Member	Use	Description
`<description>`	?	Description of the service.
`<display-name>`	?	
`<service-name>`	?	Name of the service.
`<factory-class>`		Name of the factory class for the service.
`<codebase>`	?	
`<properties>`	?	Properties needed by the service.
`<role-mapping>`	+	See "<role-mapping>" (page 117).

<services>

This tag encloses services used by the container. Table 6-56 describes its members.

Table 6-56 Members of the <services> tag

Member	Use	Description
`<security-service>`		Description of the services.
`<transaction-service>`	*	See "<transaction-service>" (page 122).

<stateful-bean>

This tag defines a stateful session bean. Table 6-57 describes its members.

Table 6-57 Members of the <stateful-bean> tag

Member	Use	Description
`<description>`	?	Description for the bean.
`<display-name>`	?	
`<small-icon>`	?	
`<large-icon>`	?	
`<ejb-deployment-id>`		Name of the bean (for example, `HelloBean`).
`<home>`		Home interface (for example, `com.my.ejb.HelloHome`).
`<remote>`		Remote interface (for example, `com.my.ejb.Hello`).
`<ejb-class>`		Implementation class (for example, `com.my.ejb.HelloBean`).
`<transaction-type>`		Value: `Container` or `Bean`.
`<jndi-enc>`	?	See "<jndi-enc>" (page 109).
`<security-role-ref>`	*	See "<security-role-ref>" (page 118).

<stateful-session-container>

This tag defines a stateful session bean container and encloses the definitions of stateful session beans. Table 6-58 describes its members.

Table 6-58 Members of the <stateful-session-container> tag

Member	Use	Description
`<codebase>`	?	
`<description>`	?	Description of the container.
`<display-name>`	?	
`<container-name>`		Name for the container.
`<properties>`	?	Used to tell the container how to handle instances of stateful session beans. See "<properties>" (page 113).
`<stateful-bean>`	+	Stateful bean definitions. See "<stateful-bean>" (page 119).

<stateless-bean>

This tag defines a stateless session bean. Table 6-59 describes its members.

Table 6-59 Members of the <stateless-bean> tag

Member	Use	Description
`<description>`	?	
`<display-name>`	?	
`<small-icon>`	?	
`<large-icon>`	?	
`<ejb-deployment-id>`		Name of the bean.
`<home>`		Home interface (for example, `com.my.ejb.HelloHome`).

Table 6-59 Members of the <stateless-bean> tag

Member	Use	Description
`<remote>`		Remote interface (for example, `com.my.ejb.Hello`).
`<ejb-class>`		Implementation class (for example, `com.my.ejb.HelloBean`).
`<transaction-type>`		Value: `Container` or `Bean`.
`<jndi-enc>`	?	See "<jndi-enc>" (page 109).
`<security-role-ref>`	*	See "<security-role-ref>" (page 118).

<stateless-session-container>

This tag defines a stateless session bean container and encloses the definitions of stateless session beans. Table 6-60 describes its members.

Table 6-60 Members of the <stateless-session-container> tag

Member	Use	Description
`<codebase>`	?	
`<description>`	?	Description of the container.
`<display-name>`	?	
`<container-name>`		
`<properties>`	?	Used to tell the container how to handle instances of stateless session beans. See "<properties>" (page 113).
`<stateless-bean>`	+	Stateless bean definitions. See "<stateless-bean>" (page 120).

<transaction-service>

This tag defines a transaction service. Table 6-61 describes its members.

Table 6-61 Members of the <transaction-service> tag

Member	Use	Description
<description>	?	Description of the transaction service.
<display-name>	?	
<service-name>		Name of the transaction service.
<factory-class>		Name of the factory class for the service.
<codebase>	?	
<properties>	?	Properties needed by the service.

APPENDIX A

Document Revision History

This document has the revisions listed in Table A-1.

Table A-1 Document revision history

Date	Notes
January 2002	Reorganized Chapter 6, "Configuration Reference" (page 77), in alphabetical order.
	Added index and glossary.

APPENDIX A

Document Revision History

Glossary

bean class The bean class implements the methods defined in an enterprise bean's business methods, which are defined in the remote interface.

bean client An application or enterprise bean that makes use of an enterprise bean.

deployment descriptor XML file that describes the configuration of a Web application. It's located in the WEB-INF directory of the application's WAR file and named web.xml. See also WAR.

EJB (Enterprise JavaBeans) Specification that provides an infrastructure through which data-based components can be developed and deployed in a variety of platforms.

J2EE (Java 2 Platform, Enterprise Edition) Specification that define a platform for the development and deployment of Web applications. It defines an environment under which enterprise beans, servlets, and JSP pages can share resources and work together.

home interface The home interface defines an enterprise bean's life-cycle methods, used to create, remove, and find beans.

ORB (Object Request Broker) Facility through which a client application can locate and use distributed objects.

remote interface The remote interface defines an enterprise bean's business methods, which are used by its clients to interact with the bean.

Web application, Web app File structure that contains servlets, JSP pages, HTML documents and other resources. This structure can be deployed on any servlet-enabled HTTP server. See also servlet container.

Index

B

C

D

E, F

G

H

I

J, K

L

M, N

O

P, Q

R

S

T, U, V

W, X, Y, Z