Create EJB components using UML modeling tools

Time required

To complete this tutorial, you will need approximately **two hours**. If you decide to explore other facets of creating entity beans using UML modeling tools while working on the tutorial, it could take longer to finish.

Prerequisites

In order to complete this tutorial end to end, you should be familiar with the following subjects:

- Java programming
- UML modeling

It will also help if you understand basic concepts about these subjects:

- Relational databases
- Container-managed entity beans
- J2EE architecture

Learning objectives

This tutorial is divided into several exercises that must be completed in sequence for the tutorial to work properly. This tutorial explains basic concepts about creating container-managed entity beans using UML class diagrams. In addition, you will learn how to map enterprise bean fields into relational database definitions, which are used to create the database tables required when the EJB application is put into production. This tutorial does not cover the design and creation of a complete enterprise application, but there are several examples provided in the Samples Gallery (**Help > Samples Gallery**). This tutorial targets beginners who have never created an EJB component, used UML class diagrams nor used the workbench. Small code snippets will be used to illustrate concepts.

This tutorial consists of these exercises:

- Exercise 1.1 introduces you to the technologies.
- Exercise 1.2 shows you how to set up the required projects in the workbench and how to create a UML class diagram.
- Exercise 1.3 shows you how to populate the class diagram with CMP entity beans and how to define relationships between classes.
- Exercise 1.4 explains top-down database mapping.
- Exercise 1.5 explains the next steps you would take in the EJB creation process.

When you are ready, begin Exercise 1.1: Introduction to the technologies

Exercise 1.1: Introduction to the technologies

An enterprise bean is a set of Java™ classes and interfaces that implement the Enterprise JavaBeans™ (EJB) specification. The EJB specification is a part of the Java 2 Platform, Enterprise Edition (J2EE) specification. J2EE is a set of standardized Java technologies that extend the Java 2 Standardized Platform, Standard Edition (J2SE). Here are a few of the technologies that J2EE includes:

- Servlets and JavaServer Pages (JSP), for handling Web browser requests and responses
- Enterprise beans (EJB beans), for handling database transactions
- Java Database Connectivity (JDBC), for database access
- Java Messaging Service (JMS), for messaging

Applications that are written using the J2EE specifications are easily deployed on any server that provides J2EE support, such as IBM®'s WebSphere® Application Server.

The most recent version of the EJB specification is 2.1. The workbench supports multiple versions of the EJB specification including 1.1, 2.0 and 2.1. You will see references to the different specification levels in the workbench. For instance, when creating a new container-managed (CMP) entity bean, you can select the CMP Version 1.x or 2.x, which correspond to the EJB versions 1.1 or 2.0/2.1, respectively.

For more information on the J2EE and EJB technologies, consult these Web sites:

- J2EE Technologies at java.sun.com
- Enterprise JavaBeans Technology at java.sun.com

Information on using these technologies can also be found in the following topics in the information center:

- EJB architecture
- J2EE architecture

Enterprise JavaBeans

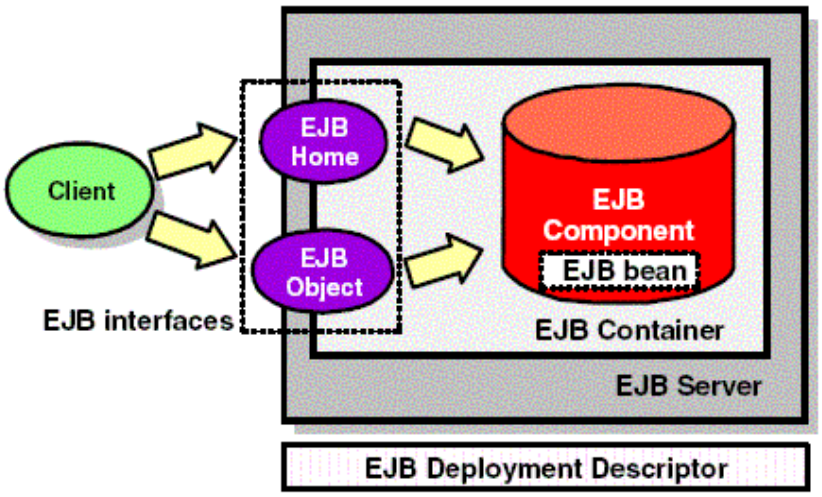
The EJB specification defines these things:

- A specific set of classes and interfaces
- The directory structure and packaging for a deployable enterprise application
- Deployment descriptors and other metadata that define the components, interactions, and behavior of the parts of the enterprise application, including information on database connectivity

An enterprise application can be installed and run on an *EJB server* (also called an Enterprise Java Server), such as the one provided by WebSphere Application Server. The server's task is to provide an *EJB container* where the enterprise beans run. An EJB server and container work together to provide these services:

- Transaction support
- Persistence of data
- Security
- Life cycle management
- Naming service
- Concurrency management
- Messaging

Client applications do not access enterprise beans directly. Instead, interfaces are provided that list the methods of the enterprise bean that are available to the client. The container provides the implementation of the interfaces in the enterprise bean.



The types of enterprise beans

There are three types of enterprise beans: *session beans*, *entity beans*, and *message-driven beans*. A typical EJB application consists of several entity beans, plus session or message-driven beans.

Entity beans are used to represent and interact with relational database tables. An entity bean typically represents a row in a database table, with the columns of the table corresponding to the fields in the bean. Data *persistence* refers to storing data permanently, in a database or other format. EJB containers can handle the persistence for an entity bean (*container-managed persistence*, or CMP); or, the entity bean can handle its own persistence (*bean-managed persistence*, or BMP).

Session beans are used to interact with entity beans. They usually provide the overall process or workflow for a client application. Session beans can be *stateless* (no data is saved) or *stateful* (some data is saved). Typically, an application client interacts with session beans, which in turn interact with entity beans.

Message-driven beans provide a communication vehicle for applications that want to access entity beans. They can be used in the same way that session beans are. The primary difference is how they are invoked. Session beans (and entity beans) are called synchronously, while message-driven beans are called asynchronously. A synchronous call to a session bean would be to invoke the session bean directly. An asynchronous call to a message-driven bean would be to send a message, for which the message-driven bean is listening. Java Messaging Service (JMS) is the underlying communications vehicle for message-driven beans.

More information on enterprise bean types can be found in the information center under EJB architecture.

The anatomy of an enterprise bean

An enterprise bean contains the following classes and interfaces:

- EJB remote component interface, or *EJB object*
- EJB local component interface, or *EJB local object*
- EJB home interface, or *EJB home*
- EJB bean class
- Primary key class

Local and remote component interfaces

The local and remote component interfaces are used by client applications to access enterprise beans. These interfaces list the available business logic methods in an enterprise bean. An enterprise bean can have a remote interface, a local interface or both.

Performance is better when you access an enterprise bean using the local interface rather than the remote interface. Some of the reasons for the performance improvement include:

- Method arguments are passed by reference using the local interface, but are passed by value using the remote interface
- Serialization of data is required to transfer data over a network when using the remote interface, while no serialization is needed using the local interface
- Network overhead occurs when using the remote interface, but there is none when using the local interface

The local interface was added to the EJB 2.0 specification to improve performance. A recommended design practice is to wrap entity beans in a *session facade*, so that clients access the session bean remotely, but the session bean accesses the entity bean locally. This practice allows remote access from clients while providing better performance.

Local and remote home interfaces

The home interface of an entity bean is also used by client applications to access the enterprise bean, but the methods available through this interface are life cycle methods, including methods to find, create, and remove entity beans within the EJB container. Home interfaces can also be remote or local.

Enterprise bean classes

Enterprise bean classes contain the business logic of the enterprise application. Methods in the interfaces expose the corresponding methods in the bean class to client applications.

Primary key classes

A *primary key* is a unique ID that is associated with a specific entry in a database. For example, if you have a database of employees, each employee has a unique employee ID. Each instance of a primary key class corresponds to one of the unique IDs, and thus to a specific employee record in the database.

The anatomy of an enterprise application

The J2EE specification defines the files and directory structure of an enterprise application. In addition, it specifies the filetypes of .war, .ear and .jar, each type used for specific purposes.

An EAR file (Enterprise Archive) is the package type for an enterprise application. It contains WAR files and JAR files, as well as an application deployment descriptor (application.xml) that contains metadata about the enterprise application.

A WAR file (Web Archive) contains files for a Web application, such as images, HTML files, servlets and JSPs. It also contains metadata in the form of a Web deployment descriptor (web.xml).

JAR files (Java Archives) contain Java classes. In EJB applications, JAR files can contain EJB modules, or they can contain an EJB client application.

EJB modules contain the enterprise beans themselves, plus metadata including an EJB deployment descriptor (ejb-jar.xml). EJB modules deployed to WebSphere Application Server can also contain metadata describing IBM-specific extensions and binding information. An EJB client application contains the Java programs used to access an EJB application, plus a client deployment descriptor (application-client.xml).

Workbench projects and file types for EJB development

File type	Meaning	Application type	Deployment descriptor
.ear	Enterprise Archive	enterprise	application.xml
.war	Web Archive	Web	web.xml
.jar	Java Archive	EJB EJB client	ejb-jar.xml application-client.xml

Using UML with Visual Editor

Visual Editor uses *Unified Modeling Language* (UML) to represent the structure and design of Java classes and interfaces, including EJB components, visually. It provides mechanisms for showing relationships between classes, as well as workflow. Using the UML tools within the workbench, you can visually edit Java classes and interfaces, or other EJB components. You can initiate the creation of an EJB component directly from a class diagram. The underlying code is generated and then visually rendered on the class diagram, ready to be edited. Changes made to a UML class diagram result in changes to the underlying code. Changes to the underlying code are reflected in the UML class diagram.

In the workbench, a UML class diagram is stored as a file with a .dnx extension.

UML 2 is a standard specification provided by the Object Management Group (OMG). The OMG Web site also provides an Introduction to UML.

The information center provides in-depth information on these topics:

- Managing UML diagrams.
- Visually developing EJB applications with UML diagrams.

Now you are ready to begin Exercise 1.2: Preparing the workspace.

Exercise 1.2: Preparing the workspace

Before you begin, you might want to review the concepts in Exercise 1.1: Introduction to the technologies.

Exercise 1.2 gives an overview of the workbench layout and EJB-related projects. In this exercise, you will do these tasks:

- Create an enterprise application (EAR) project
- Create an EJB project
- Create an EJB client project
- Create a UML class diagram

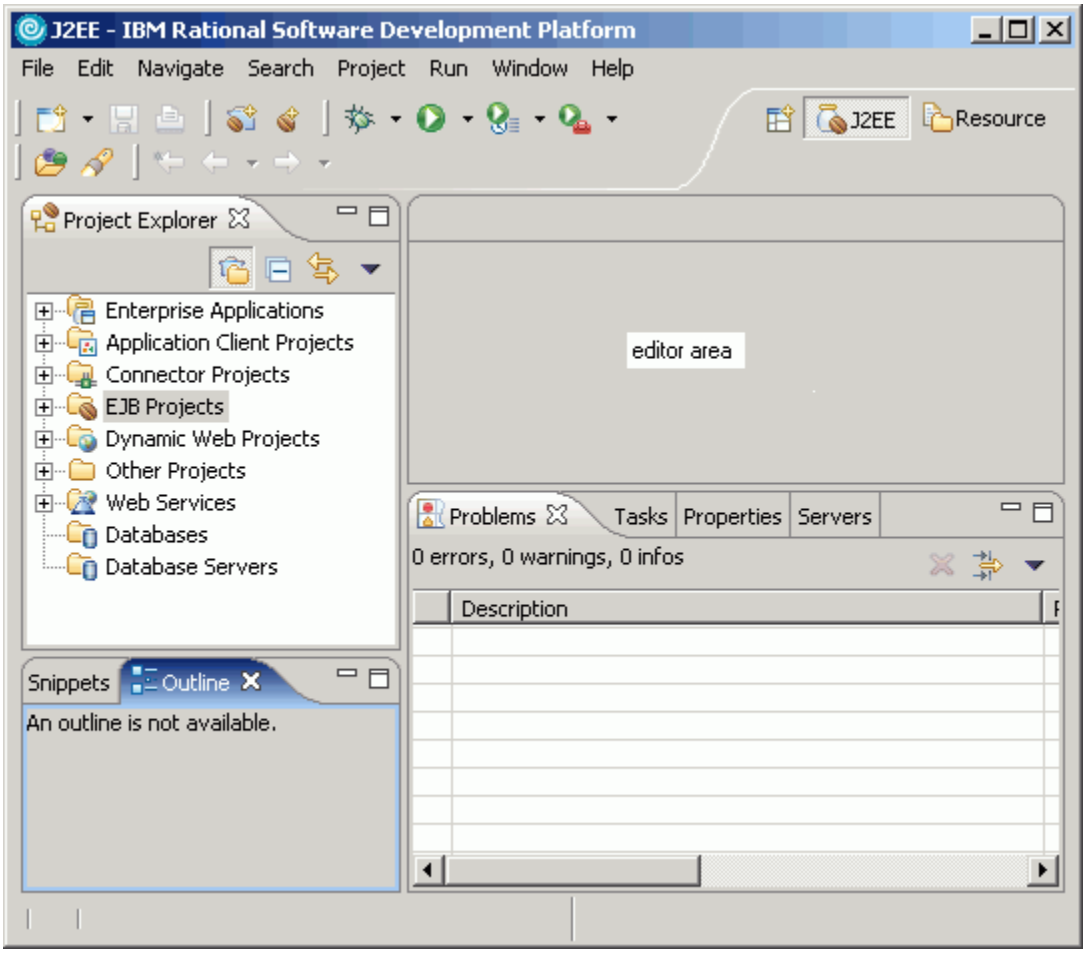
Workbench layout

All work done in the workbench must be associated with a *project*. Projects provide an organized view of the work files and directories, optimized with functions based on the type of project.

The layout of the workbench consists of *perspectives* and *views*. Views are panes in the workbench providing different ways of looking at the data. For instance, projects are displayed in a *Project Explorer* view. The structure of a class or XML file would be displayed in an *Outline* view. The workbench has many different views, specialized for each type of data. In a given area of the workbench, views may be stacked, so that they appear as tabbed pages for easy access.

A perspective manages the overall layout of the workbench, and comprises a set of views and an *editor area*. Perspectives come with a set of predefined or default views, laid out in a specific arrangement. However, the perspectives are highly customizable. Different views can be added or removed from a given perspective. The location of a view can be changed.

The J2EE perspective is optimized for EJB development.



The information center provides more information on these topics:

- Views
- Perspectives
- Workbench

Setting up the projects

In the workbench, all files must reside in a project, so before you create the UML class diagram, you need to create a project to put it in. Since you are creating an EJB, you will create an EJB project. When you create an EJB project, other supporting projects are automatically created.

Creating an EJB project

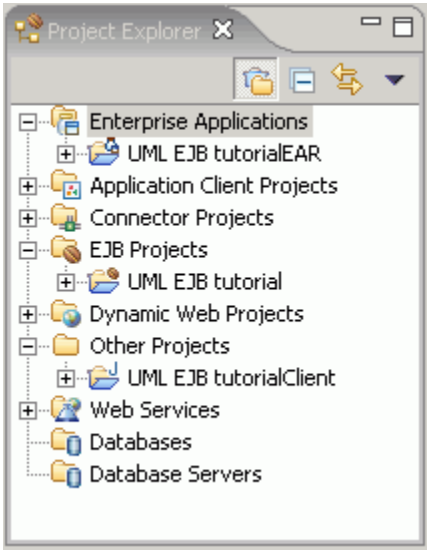
Creation of an EJB project requires that a Target server be defined. Make sure that you included the WebSphere Application Server 6.0 Integrated Test Environment during installation of this product.

1. From the **File** menu, select **New > Other**. The New wizard opens.
2. Select the **Show All Wizards** check box.
3. Expand **EJB**.
4. Select **EJB Project**.
5. Click **Next**.
6. If the Confirm Enablement dialog box opens, click **OK**.
7. When the New EJB Project wizard opens, give the project a name (`UML EJB tutorial` in this example)
8. Leave the project location as it is.
9. If the Advanced options are displayed, there is no Target server defined. If this occurs, you need to

do one of these steps:
    - Rerun the product installation program and install the WebSphere Application Server 6.0 Integrated Test Environment.
    - Define a Target server by selecting **New** and defining a locally installed version of WebSphere Application Server.
10. Click **Finish**.
11. If the Confirm Perspective Switch dialog box opens, answer **Yes**. This switches you to the J2EE perspective.

The EJB project is created along with several supporting projects, as shown in the Project Explorer view of the J2EE perspective:

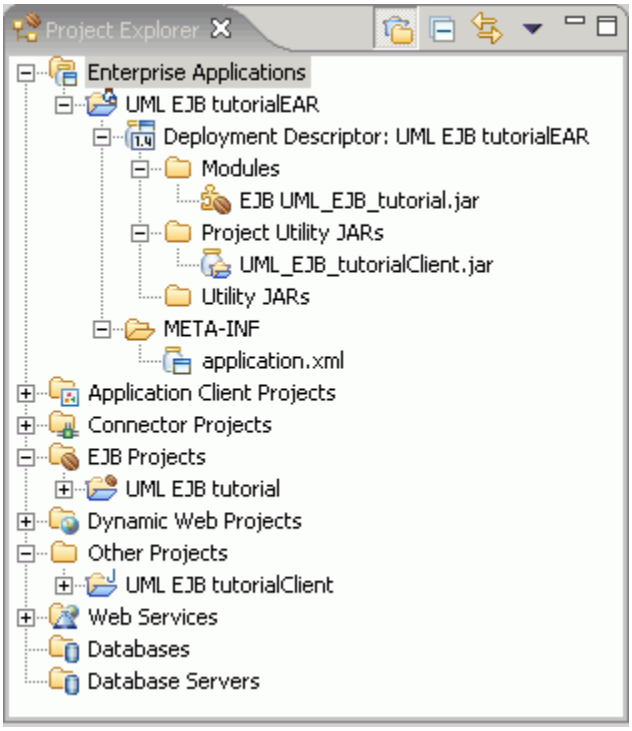
- The EAR project `UML EJB tutorialEAR` in Enterprise Applications
- The EJB project `UML EJB tutorial` in EJB Projects
- The EJB client project `UML EJB tutorialClient` in Other Projects



The Enterprise Application (EAR) project

In the Project Explorer view, expand **Enterprise Applications** and explore the `UML EJB tutorialEAR` project. Within the EAR directory you will see

- The `Deployment Descriptor` directory that contains three subdirectories. The Deployment Descriptor directory is also a link to the application.xml file in the META-INF directory.
    - The `Modules` directory containing `EJB UML_EJB_tutorial.jar`. This is the primary JAR file for the enterprise beans. It is associated with the EJB project.
    - The `Project Utility JARs` directory containing `UML_EJB_tutorialClient.jar`. This JAR file contains the code needed by an EJB client. It is associated with the EJB client project.
    - The `Utility JARs` directory. This directory can hold other classes that are used by the EJB components.
- The `META-INF` directory, containing the application.xml deployment descriptor.



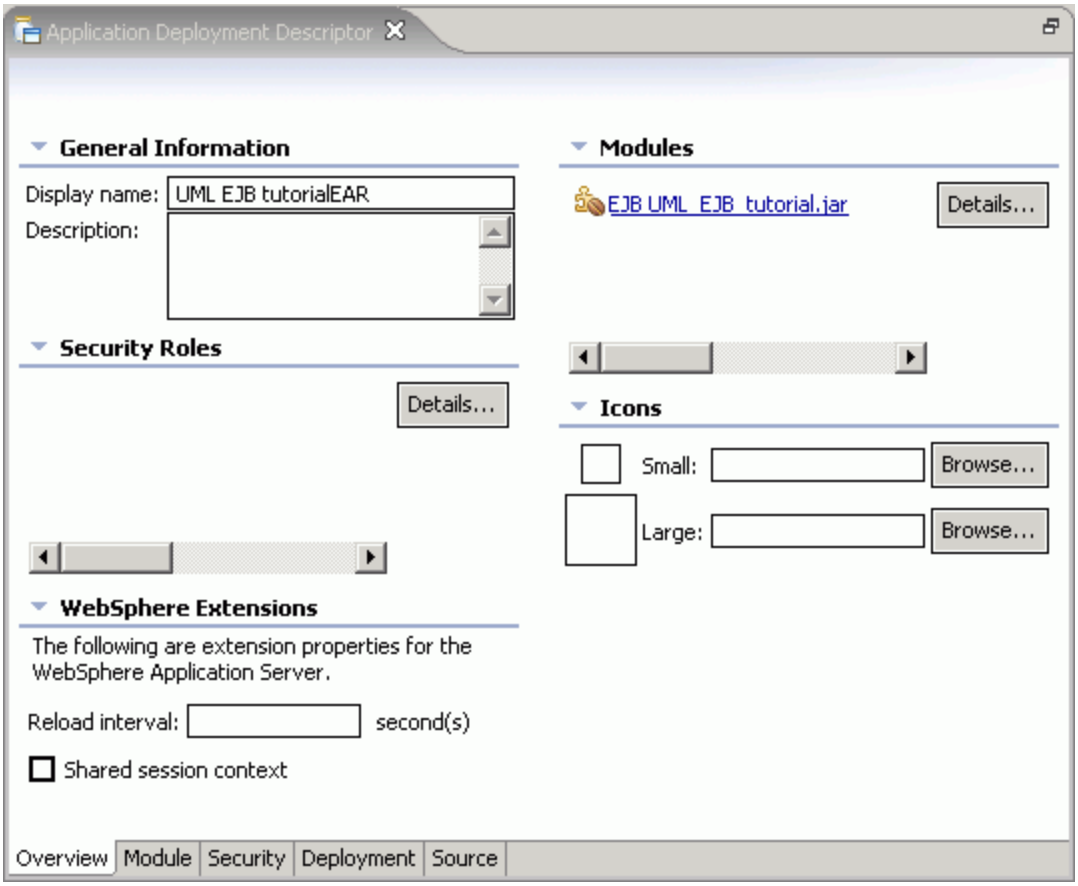
If you double-click the deployment descriptor (either the link or the actual file), the application.xml file opens in the Application Deployment Descriptor editor. Contents of the application deployment descriptor are shown on several different pages and can be navigated by clicking the tabs at the bottom of the editor area.

You can double-click the title bar (

) of the Application Deployment Descriptor editor to expand it to use the entire workbench frame. Double-click the title bar again when you want to restore the window to its normal size.

Notice on the Overview page, under the General Information heading, the Display name is `UML EJB tutorialEAR`, which is the same as the EAR project name. This name is derived from your EJB project name. Display name is the value seen when installing and configuring an EAR file in the WebSphere Application Server Administrative console. Of course, you can choose your own names when creating the EJB project, or rename them after creation.

Also note, under the Modules heading, `EJB UML_EJB_tutorial.jar`. On the Module tab, in addition to this EJB JAR file, you will see the Project Utility JAR file, `UML_EJB_tutorialClient.jar` listed.

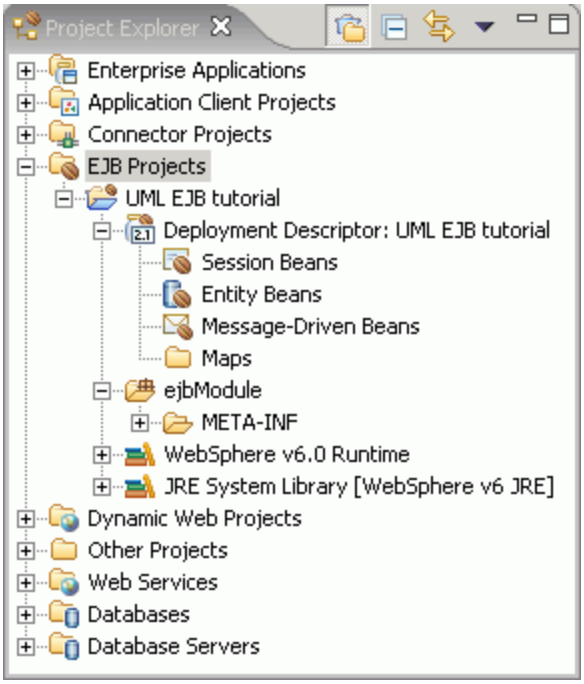


Close the deployment descriptor editor by clicking the **X** on the title bar.

The EJB project

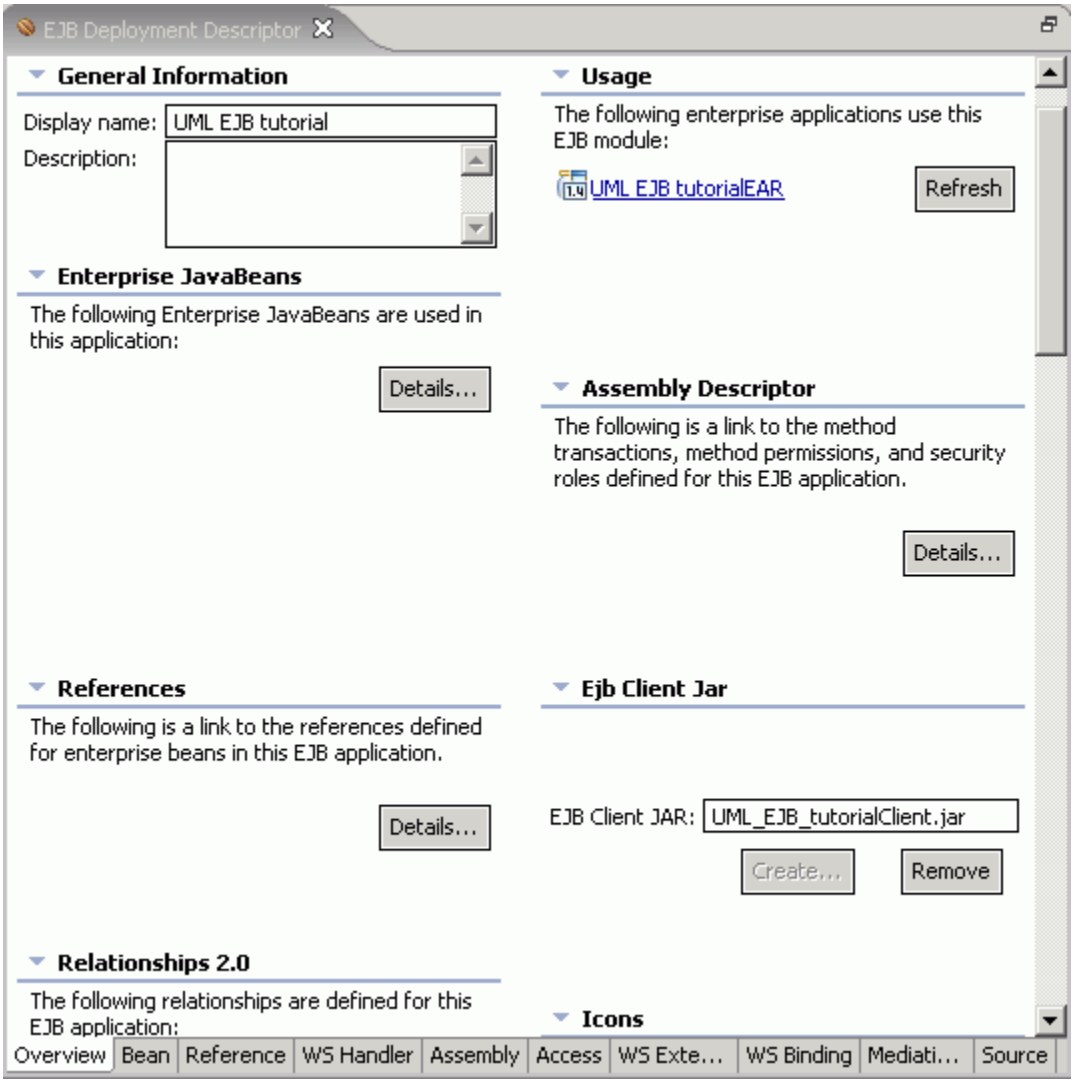
In the Project Explorer view, collapse all open projects ( ). Expand **EJB Projects**, then **UML EJB tutorial**. You will find these resources:

- The `Deployment Descriptor` directory, that contains four subdirectories. The Deployment Descriptor directory is also a link to the ejb-jar.xml file in the ejbModule > META-INF directory.
  - The `Session Beans` directory that will list session beans associated with the EJB project.
  - The `Entity Beans` directory that will list entity beans associated with the EJB project.
  - The `Message-Driven Beans` directory that will list message-driven beans associated with the EJB project.
  - The `Maps` directory that will list mapping relationships between the enterprise beans and the relational database tables.
- The `ejbModule` directory that contains the `META-INF` directory. The META-INF directory contains the ejb-jar.xml deployment descriptor.
- The `WebSphere v6.0 Runtime` directory, containing code libraries shipped with WebSphere Application Server v6.0.
- The `Java Runtime Environment (JRE) System Library [WebSphere v6 JRE]` directory, which are the Java libraries shipped with WebSphere Application Server v6.

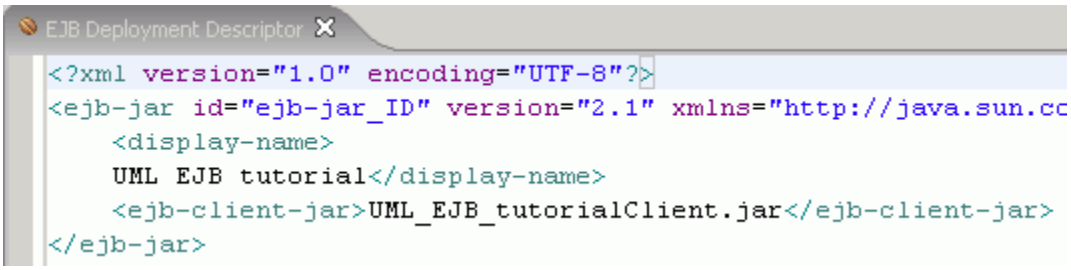


If you double-click the deployment descriptor (either the link or the actual file), the ejb-jar.xml file opens in the EJB Deployment Descriptor editor. Contents of the EJB deployment descriptor are shown on several different pages and can be navigated by clicking the tabs at the bottom of the editor area.

Notice on the Overview page, under the General Information heading, the Display name is `UML EJB tutorial`. This is the same as your EJB project name. Under the Usage heading, you will see the `UML EJB tutorialEAR`, indicating the EAR file that uses this JAR file. Under the EJB Client Jar heading, you will see the `UML_EJB_tutorialClient.jar` listed.



Click on the Source tab of the deployment descriptor. This view shows the actual XML data that is displayed on the other pages of the deployment descriptor editor. When changes are made to the enterprise application, via the deployment descriptor editor or via the UML class diagram, the changes are made to the XML data, then shown in the deployment descriptor editor and the class diagram.



There are many other tabs for the EJB deployment descriptor editor, but these details are beyond the scope of this tutorial. See the EJB deployment descriptor editor section of the information center for more information.

Close the deployment descriptor editor by clicking the **X** on the title bar.

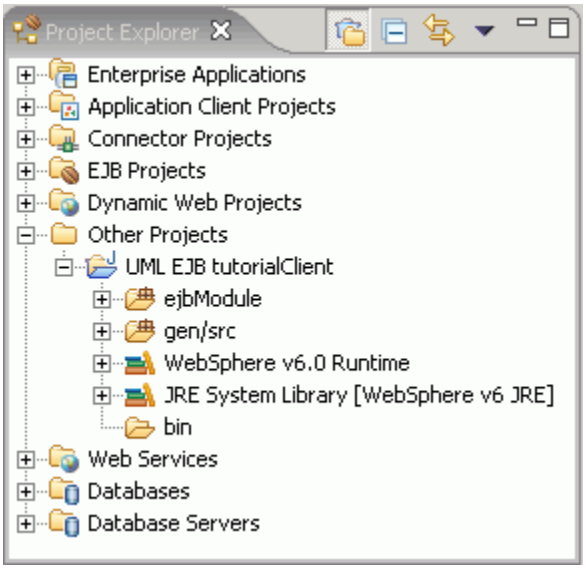
The EJB Client project

In the Project Explorer view, collapse the expanded projects again ( ).

Expand **Other Projects**, then **UML EJB tutorialClient**. You will find these resources:

- The `ejbModule` directory, which will contain the local and local home classes.
- The `gen/src` directory, where generated code is placed. For instance, if you generate a session facade bean, the code is placed here, as well as the gen/src directory in the EJB project.
- The `WebSphere v6.0 Runtime` directory, the same as in the EJB project.
- The `JRE System Library [WebSphere v6 JRE]` directory, the same as in the EJB project.
- The `bin` directory, which will eventually contain compiled Java classes.

Notice that there is no deployment descriptor for this type of project.



Creating the UML class diagram

Now that you have your projects defined, you will create a UML class diagram. You will create a folder to contain the class diagram, to keep it separate from the actual code.

In the Project Explorer view, do these steps:

1. Highlight the EJB project **UML EJB tutorial**.
2. Right-click, select **New > Other**.
3. Expand **Simple**.
4. Select **Folder**.
5. Click **Next**.
6. Verify that the parent folder name is `UML EJB tutorial`.
7. Provide the folder name `diagrams`.
8. Click **Finish**.

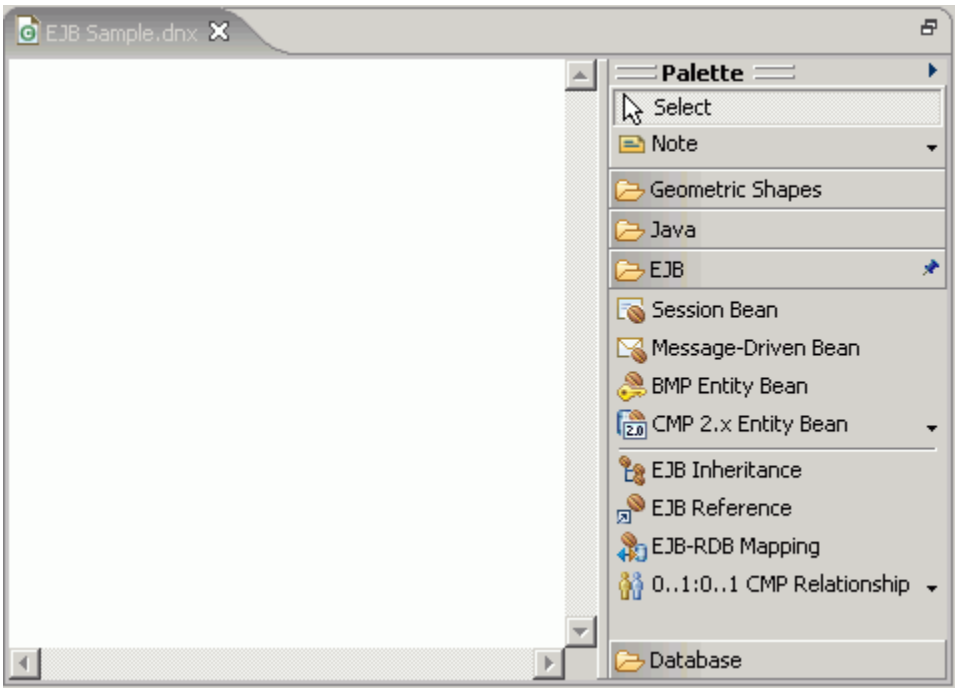
Now, create an empty UML class diagram by following these steps:

1. Highlight the EJB project **UML EJB tutorial**.
2. Right-click and select **New > Class Diagram**. The New Class Diagram wizard opens.
3. Expand **UML EJB tutorial**.
4. Select the **diagrams** folder.

5. Change the class diagram file name to `EJB Sample`
6. Click **Finish**.

This creates an empty class diagram and opens the empty file in the UML visual editor. Notice the palette on the right side of the class diagram. The palette contains items that can be created, visualized, and edited on the class diagram. In general, click on any palette object to highlight it, then click anywhere in the class diagram to drop the item onto the class diagram. The object will appear on the class diagram after any underlying code to support that object is created in your project. If creation of the object involves a wizard, the wizard is automatically launched.

Notice the EJB drawer in the palette. Since our class diagram is inside an EJB project, the EJB drawer is automatically included on the palette.



Now you are ready to begin Exercise 1.3: Adding entity beans to class diagrams.

Exercise 1.3: Adding entity beans to class diagrams

Before you begin, you must complete Exercise 1.2: Preparing the workspace.

Exercise 1.3 guides you through the creation of entity beans from a UML class diagram. In this module, you will do these tasks:

- Add entity beans to the class diagram
- Show relationships between the beans
- Explore additional useful views
- Manipulate objects on the class diagram

Description of the beans

You will create two entity beans in this scenario: Author and Book. The Author bean will represent a database table, also named AUTHOR. The bean will contain fields representing the author: ID, name, age and gender. These fields in the bean will map directly to columns in the AUTHOR table.

The Book bean will represent a book. Its fields include ID and title. Data corresponding to the Book bean will be stored in the BOOK table.

Creating the Author bean

Follow these steps to create the Author bean:

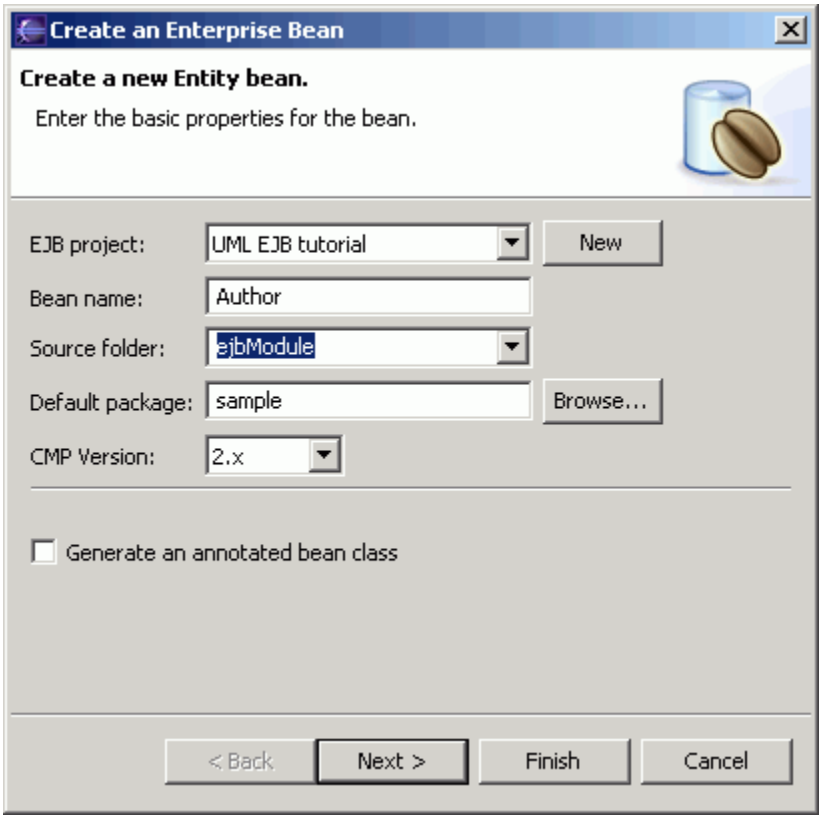
1. Locate the EJB drawer of the palette on the right of the UML class diagram.
2. Click on the EJB drawer to open it if it is not already open. `CMP 2.x Entity bean` appears in the middle of the list.
3. Click the arrow ( ) to the right of `CMP 2.x Entity bean` to reveal more choices. The other choice listed is `CMP 1.x Entity bean`.
4. Select the `CMP 2.x Entity bean`.
5. Move the mouse pointer onto the class diagram and click where you want the bean placed. This creates a new CMP 2.x entity bean and visualize it in the class diagram.

Since EJB beans are complex, this action launches the Create an Enterprise Bean wizard, allowing you to define the bean.

The Create an Enterprise Bean page

On the first page of the wizard, follow these steps:

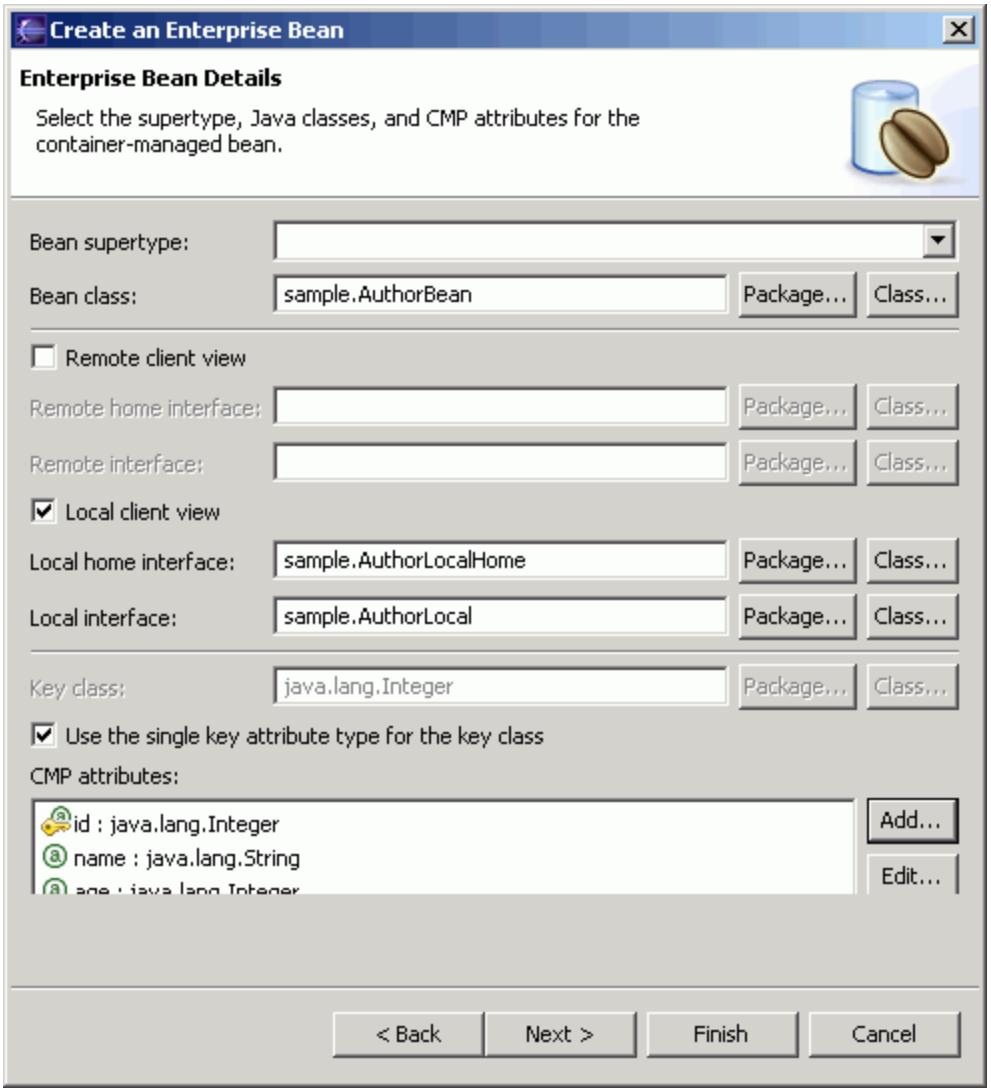
1. Verify that the **EJB project** is `UML EJB tutorial`.
2. Provide the **Bean name**, `Author`
3. Verify the **Source folder** is `ejbModule`.
4. Change the **Default package** name to `sample`
5. Verify that the **CMP Version** is `2.x`.
6. Click **Next**.



The Enterprise Bean Details page

On the second page of the wizard, follow these steps:

1. Leave the **Bean supertype** blank.
2. Verify that **Bean class** is `sample.AuthorBean`.
3. Leave the **Remote client view** check box cleared.
4. Select the **Local client view** check box.
5. Verify the **Local home interface** name is `sample.AuthorLocalHome`.
6. Verify the **Local interface** name is `sample.AuthorLocal`.
7. Verify the **Key class** is `java.lang.Integer`.
8. Select the **Use the single key attribute type for the key class** check box.
9. Verify that the key `id` is on the list of CMP attributes.
10. Add two CMP attributes:
    a. Click **Add**.
    b. Create an attribute named `name` with a type of `java.lang.String`. Leave the **Array** and **Key field** check boxes cleared and the **Promote getter and setter methods to local interface** check box selected. When getters and setters are promoted to the local interface, they are exposed to client applications.
    c. Click **Apply**.
    d. Create an attribute named `age` with a type of `java.lang.Integer`. Leave the **Array** and **Key field** check boxes cleared and the **Promote getter and setter methods to local interface** check box selected.
    e. Click **Apply**, then **Close**.
    f. Click **Finish**.



Your first entity bean will be created and visualized on the class diagram. The following files appear in the Project Explorer view:

- In the EJB project under **ejbModule > sample**
  - AuthorBean.java -- the bean
- In the EJB client project under **ejbModule > sample**
  - AuthorLocal.java -- the local interface
  - AuthorLocalHome.java -- the local home interface

Open the EJB deployment descriptor and notice that on the Overview page, under the Enterprise JavaBeans heading, the Author bean is listed. Click on the Author bean entry. The Bean page of the deployment descriptor editor opens, showing the three CMP fields and the four class and interface files, including the class used by the primary key, java.lang.Integer.

On the Bean page, under the WebSphere Bindings heading, the JNDI name `ejb/sample/AuthorLocalHome` is shown. This JNDI name is used when a client application is searching for the bean.

In the Project Explorer view, under the EJB project, the **ejbModule > META-INF** directory contains the new file `ibm-ejb-jar-bnd.xmi`. This file contains IBM-specific binding information. The information within it is accessible through the deployment descriptor editor, but the values are stored a separate file to allow you easily remove IBM-specific enhancements if you want to deploy your application on another vendor's EJB

server. Later in the EJB development process, the file `ibm-ejb-jar-ext.xmi` will appear. It contains IBM-specific extension information.

Also in the Project Explorer view, expand the deployment descriptor directory under the EJB project, expand **Entity Beans > Author** and explore the links to the various components that comprise the Author bean.

Creating the Book bean

Next, you will create the Book bean. Follow these steps:

1. Add another CMP 2.x entity bean to the class diagram.
2. Follow the same steps as for the Author bean, with these differences:
    - The bean name is `Book`
    - These are the CMP attributes:
        - `id`, of type `java.lang.Integer` (the `id` attribute is created for you automatically)
        - `title`, of type `java.lang.String`
3. Click **Finish**.

Your second entity bean is created and visualized on the class diagram and the following files appear in the Project Explorer view:

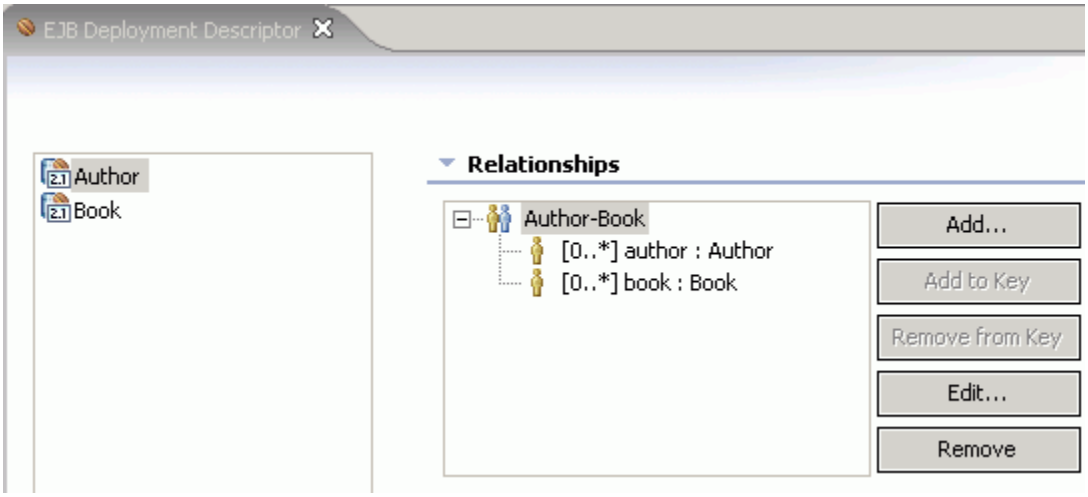
- In the EJB project under **ejbModule > sample**
    - BookBean.java -- the bean
- In the EJB client project under **ejbModule > sample**
    - BookLocal.java -- the local interface
    - BookLocalHome.java -- the local home interface

The EJB deployment descriptor now has Book on the list of Enterprise JavaBeans on the Overview page. The Book bean has details similar to the Author bean. From the Project Explorer view, expand the deployment descriptor directory under the EJB project. Expand **Entity Beans > Book** and explore the links to the bean's components.

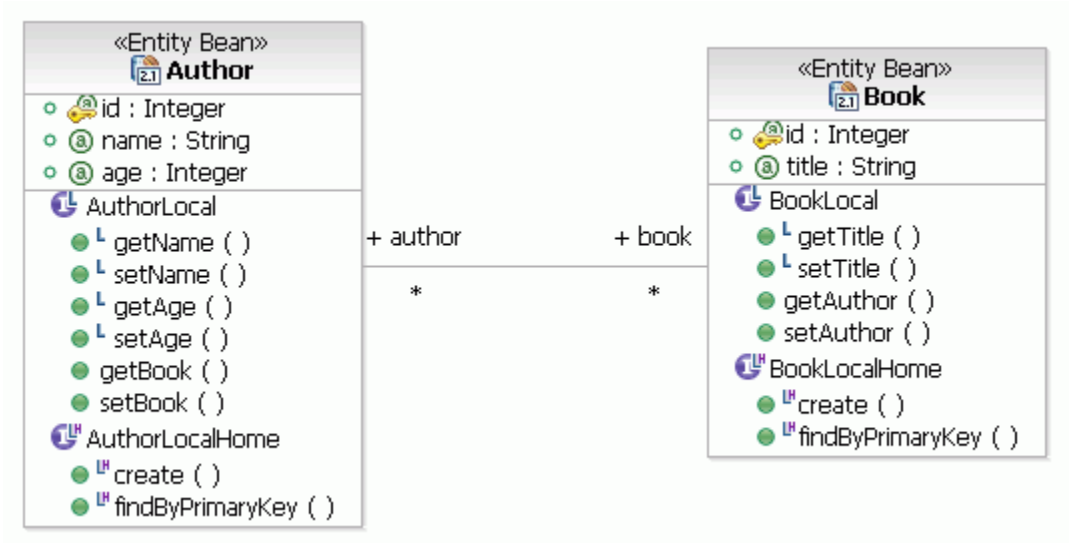
Defining relationships between beans

Next, you will define a CMP relationship between the two CMP beans on the class diagram. A single Author can be associated with zero or more books. And a single book can be associated with one or more authors. This relationship would be expressed as a *bidirectional many-to-many relationship* using the notation `0..*:0..*`.

1. Double-click the title bar of the UML class diagram to enlarge the window.
2. In the palette view, expand the **CMP relationship** drawer by clicking the arrow ( ).
3. Select the **0..*:0..* CMP relationship**.
4. Move the mouse pointer over the Author bean. You will see an arrow icon () if the relationship is valid for the bean. If the relationship is not valid, you will see the same icon plus an invalid symbol () .
5. Click on the Author bean.
6. Drag the cursor to the Book bean and release it. If this target bean is invalid for the relationship, the invalid icon () appears.
7. The CMP relationship is visualized as an *association* on the diagram after the appropriate underlying code is generated. In the deployment descriptor, these relationships appear under the Relationships heading on the Bean page.
8. Select **Ctrl + S** to save the class diagram.



You may want to rearrange elements in the class diagram to see the details.

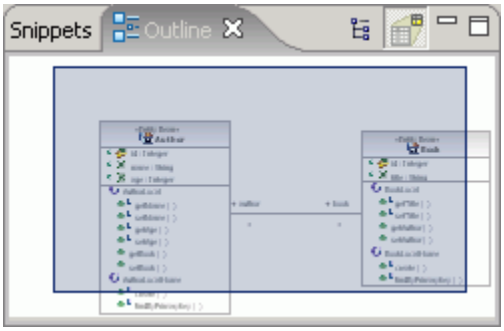


Exploring additional views

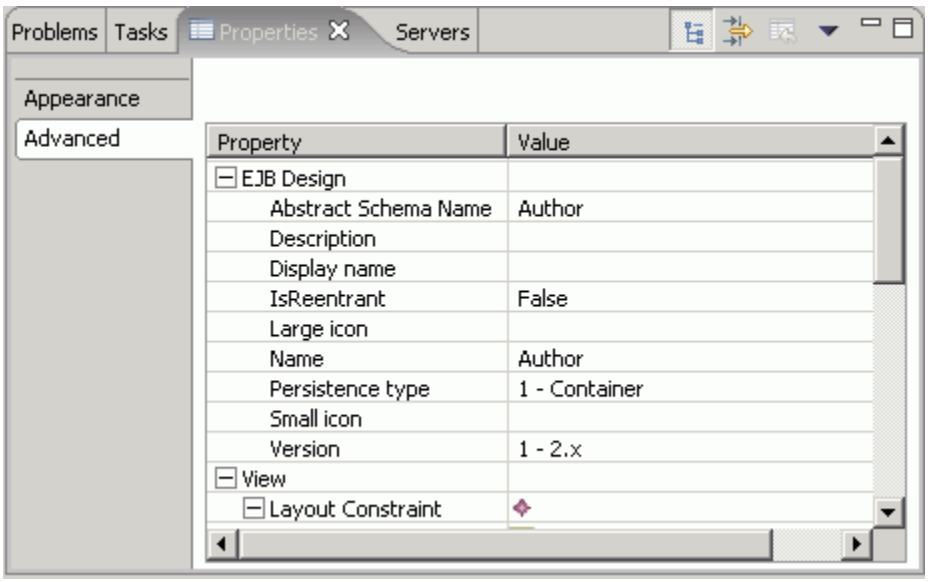
The Outline and Properties views provide useful information on the overall class diagram and on specific EJB component properties.

The Outline view appears, by default, in the bottom left of the workbench. Use this view to navigate a large diagram following these steps:

1. If the class diagram is still using the entire workbench frame, double-click the title bar of the class diagram again to restore it to normal size.
2. Notice the Outline view in the bottom left corner of the workbench. The class diagram is displayed there on a small scale. If the diagram is too large to see completely in the edit window, you will see a highlighted rectangle in the Outline view which indicates the visible portion of the diagram.
3. In the Outline view, move the highlighted rectangle around with your mouse pointer. The section of the class diagram that is visible in the editor area moves to correspond to the highlighted section in the Outline view.



The Properties view appears, by default, in the bottom right of the workbench. It has two tabs, Appearance and Advanced. You can change colors, fonts and other visual properties on the Appearance page. The Advanced page shows detailed properties of the highlighted element in the class diagram. Some of the property values in the Properties view are read-only and some can be modified.



Modifying attributes from the UML class diagram

Next, you will modify some of the CMP attributes for the Author bean.

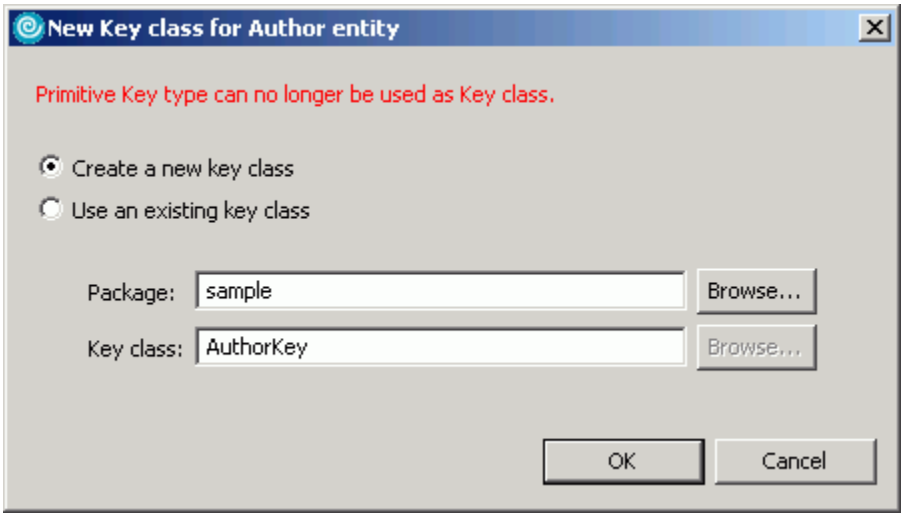
Add a CMP attribute to a bean

1. Move the mouse cursor anywhere on the Author bean. An *Action bar* appears.

2. Click the primary key symbol () in the Action bar to add additional CMP attributes to AuthorBean. The CMP Fields wizard opens.
3. Add a field with the name `gender` of type `java.lang.Character`.
4. Click **Apply** > **Close** > **Finish**. The bean classes are updated and the `gender` attribute appears in the Attribute compartment of the bean on the class diagram.

Make a CMP attribute part of the key

1. Highlight the `name` attribute in the Author bean.
2. Right-click and select **Edit > Add to Key**. The New Key class wizard opens. A new key class is now required because the key is no longer a primitive Java type. Prior to this step, the java.lang.Integer was used as the key class.
3. Click **OK** to use the preselected options to create the new key class with the default key class name, `AuthorKey`. The name attribute now appears on the class diagram with the primary key field symbol. The new key class appears in the **UML EJB tutorialClient > ejbModule > sample** directory
4. Select **Ctrl + S** to save the class diagram.



Manipulating objects in the UML class diagram

Next, look at the details of the class diagram. Double-click the title bar of the class diagram to enlarge it. Now the class diagram editor takes up most of the workbench window, leaving the palette expanded on the right.

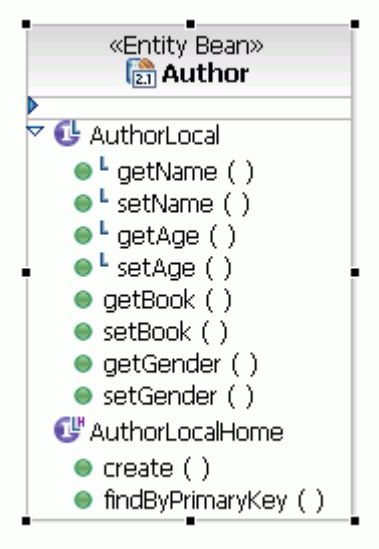
Rearrange the bean components on the diagram

1. Click once on one of the beans. Small black squares appear around the bean's rectangle, indicating that it has been selected.
2. Drag the bean to another location on the class diagram and drop it. If the Author or Book beans are moved, the relationship arrow adjusts to reflect the new position.

Change the items displayed for a bean

1. In the class diagram editor, highlight the Author bean. The rectangle representing the bean is divided into several compartments. The default compartments that are displayed are Attribute and Provided Interfaces. The compartments have a hide icon (▽) in the top left corner.
2. Click on one of the hide icons to hide that compartment of the bean. The compartment collapses and the hide icon is replaced with the show icon (▷).

3. Click on the show icon to re-expand that compartment.
4. Select which compartments are shown or hidden by right-clicking the bean and selecting **Filters > Show/Hide Compartment**.

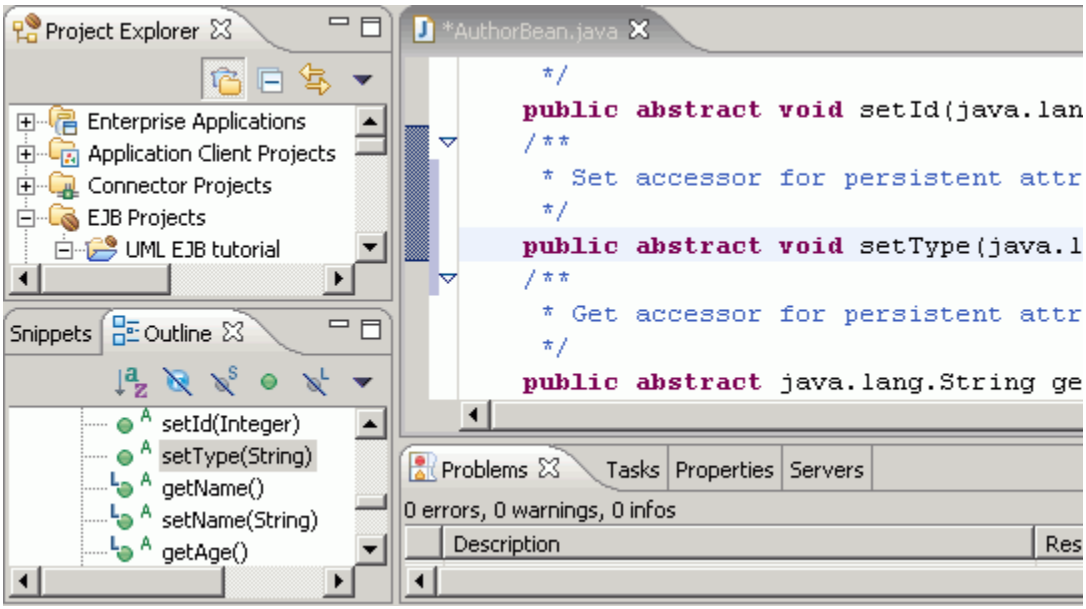


Open editors from the diagram

- Double-click the bean name or the Attribute compartment of a bean in the class diagram. The EJB deployment descriptor editor opens.
- Highlight a bean in the class diagram. Right-click and select **Navigate > Open with > EJB Bean Java Editor**. The Java class editor opens on the Java class selected.

When you open the EJB Bean Java Editor, an Outline view for this editor appears in the lower left corner of the workbench. You can make changes to the bean from this view. Right-click one of the methods in the Outline view and select **Enterprise Bean**. Select one of the Promote or Demote choices to promote or demote the selected method to one of the interface types listed.

Try adding a method to the class using the EJB Bean Java Editor. The new method appears in the Outline view. If you promote the new method to an interface, the method appears on the class diagram.



Familiarize yourself with the decorations associated with the beans

Decoration	Meaning
	CMP 2.x entity bean
	primary key field
	CMP field, not primary
	local interface
	local home interface
	remote interface
	local method
	local home method
	remote method

A complete list of decorations is shown in the information center under UML visualization reference.

UML class diagrams and code interaction

As you make changes in your class diagram, corresponding changes are automatically made to the underlying code. Conversely, if you make changes directly to the code, the class diagram is automatically altered to reflect the change. Any type of change made within the workbench results in automatic and immediate revalidation of the code. For EJB applications, validation includes identifying compilation errors in Java classes, class path problems, and violation of EJB specifications.

At the bottom of the workbench, several views are stacked, including one named Problems. This view shows all errors, warnings and informational messages for all projects and files within the workbench. Double-clicking one of the listed problems opens the appropriate file to the place where the error occurs.

The Problems view limits the number of problems listed to 100 by default. You can change this limit, or filter the view to display only items you want to see. Filtering options include problem severity, projects, resources, and problem type. To modify these default settings, click the arrow () and select **Filters**.

Configuring preferences for UML class diagrams

You can change some of the default behaviors and display characteristics of class diagrams. From the **Window** menu, select **Preferences**. Expand **Modeling** to see the items that can be changed.

The main section contains Global Settings, such as whether Action Bars appear when you hold the mouse pointer over a class in the diagram. The Appearance section has several pages that allow you to change things like fonts, colors, connectors, and shapes. The EJB section has a settings for when class diagrams open. Other sections include options for Java fields and methods, rules and grids, and validation.

Step through the sections to become familiar with the various options. After making changes, click **OK** to save them and exit the Preferences page.

Now you are ready to begin Exercise 1.4: Mapping entity beans to relational databases.

Exercise 1.4: Mapping entity beans to relational databases

Before you begin, you must complete Exercise 1.3: Adding entity beans to class diagrams.

Exercise 1.4 teaches you how to create the database mapping files for Cloudscape™. It explains the files created and their use. In this module, you will do these tasks:

- Define the target database type
- Generate relational database mapping files

Overview

Next, you will map the entity beans to a relational database. Mapping involves creating database definition files for the tables that will be used to store the entity bean's fields. Generally, each bean uses a different table. Fields in the bean represent columns in the table. Key fields in the bean correspond to primary keys in the database table.

Database mapping results in a set of files that are discussed later. These files are packaged with the EJB. They are also used by an administrator to create the database tables when putting the EJB application into production. The database mapping files are dependent on the database type.

You can create a set of database mapping files for one or several database types, which are listed below.

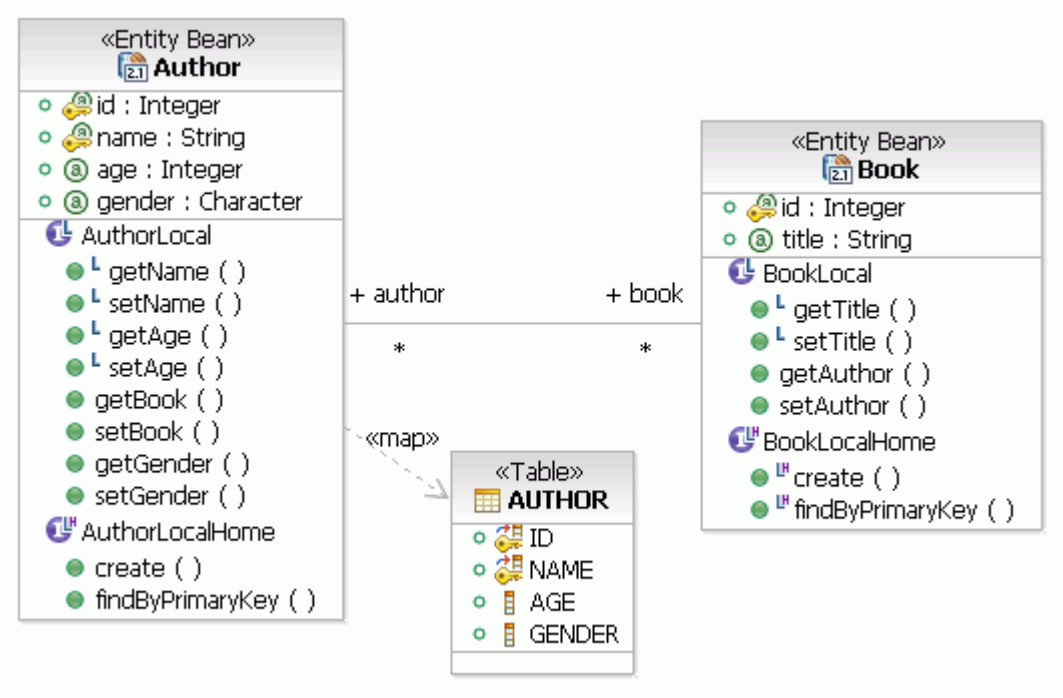
- Cloudscape (included with the workbench)
- DB2 Universal Database
- Oracle
- Informix® Dynamic Server
- Microsoft® SQL Server
- Sybase Adaptive Server Enterprise
- InstantDB
- MySQL
- Other databases that support the SQL-92 or SQL-99 standards

This approach, creating database definitions from existing entity beans, is called *top-down mapping*. The reverse process, *bottom-up mapping*, is also provided, where you create entity beans from existing database tables. *Meet-in-the-middle mapping* is also available, for use when you have existing enterprise beans and existing database tables. See Mapping enterprise beans to database tables for more information.

Mapping relational databases from entity beans

To create mapping files for Cloudscape, follow these steps:

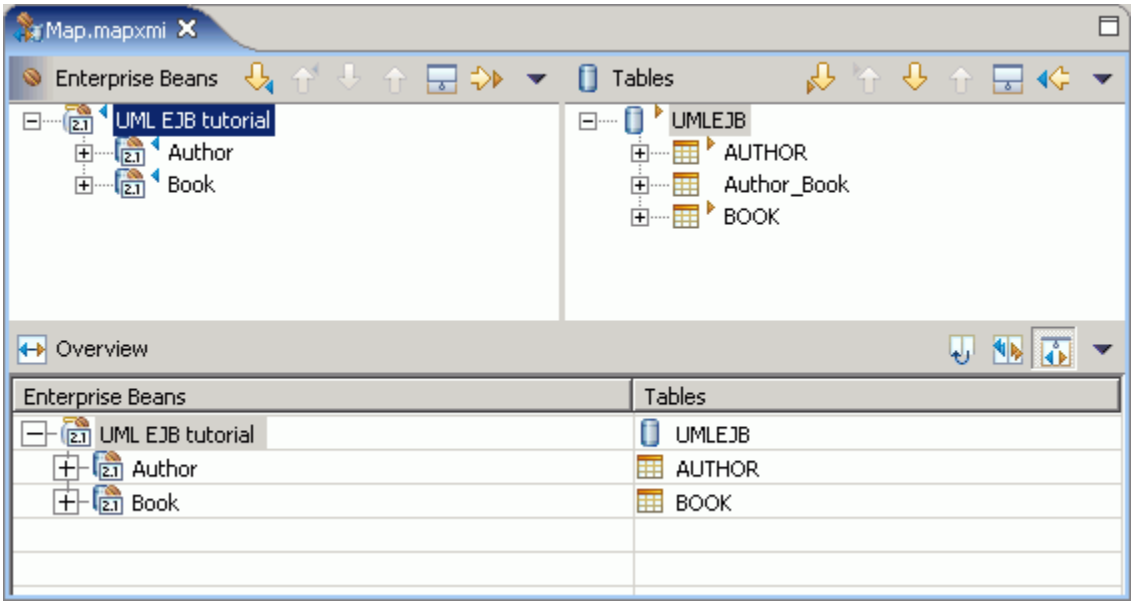
1. On the class diagram, highlight the Author bean.
2. Right-click and select **EJB to RDB Mapping > Generate Map**. The EJB to RDB Mapping wizard opens.
3. On the first page, select **Create a new backend folder**.
4. Click **Next**.
5. Select **Top-Down** as the type of mapping.
6. Click **Next**.
7. On the Top-Down Mapping Options page,
   a. Select `Cloudscape V5.1` as the **Target Database**.
   b. Use `UMLEJB` as the **Database name**.
   c. Leave `NULLID` as the **Schema name**.
   d. Leave the other options as they are.
8. Click **Finish**. The mapping files are created and the Author table is visualized on the class diagram.
9. Select **Ctrl + S** to save the class diagram.



In the Project Explorer view, under **EJB Projects > UML EJB tutorial > ejbModule > META-INF > backends**, you will see the folder **CLOUDSCAPE_V51_1** which contains the mapping files listed below. Another view of the mapping relationships can be found in **EJB Projects > UML EJB tutorial > Deployment Descriptor > Maps** directory. You can locate the newly mapped table in the Project Explorer view by selecting the table on the class diagram, right-clicking and selecting **Navigate > Show in > Data Definition View**.

- `Map.mapxmi`

  This XML file defines the relationships between the CMP fields in the beans and the database schema. Double-click the file in the Project Explorer view to see a visual representation.



- `Table.ddl`

  This is the data definition file. It contains the SQL table creation statements and other table definitions. It is used to create the actual database that will be used to persist the EJB application data on the test or production server.

- `UMLEJB_NULLID_AUTHOR.tblxmi`

  This XML file has a name derived from the database name, the schema name and the table name. It uses the XMI tag library to define entries for the database, schema, columns and data types. This particular file defines the Author bean table.

- `UMLEJB_NULLID_BOOK.tblxmi`

  This XML file defines the Book bean table.

- `UMLEJB_NULLID_Author_Book.tblxmi`

  This XML file defines the relationships between the Author and Book beans.

- `UMLEJB_NULLID.schxmi`

  This XML file defines the schema, with pointers to the other XMI-encoded files.

- `UMLEJB.dbxmi`

  This XML file defines the database, with pointers to the other XMI-encoded files.

Now you are ready to begin Exercise 1.5: Finishing the EJB creation process.

Exercise 1.5: Finishing the EJB creation process

Before you begin, you must complete Exercise 1.4: Mapping entity beans to relational databases.

Exercise 1.5 explains the next steps. In this module, you will be introduced to these concepts:

- Deploying an enterprise application
- Exporting an EAR file
- Creating database tables
- Testing the enterprise application
- Installing the enterprise application

The next steps

This tutorial has shown you the techniques for using UML class diagrams to create some simple CMP entity beans. However, the files created do not comprise a complete enterprise application. Some examples of enterprise applications are in the Samples Gallery (**Help > Samples Gallery**). A real application would also have these elements:

- Business logic added to the skeleton methods of the classes.
- Session or message-driven beans to drive the overall program logic.
- Optionally, a Web module containing related HTML files, servlets and JSPs.

For this discussion, assume that you have a complete enterprise application. The next steps in the process are as follows:

1. Deploy the enterprise application. The deployment step will package the projects into the JAR, WAR and EAR files defined by the J2EE specification. To deploy an enterprise application, follow these steps:
    a. From the Project Explorer view, highlight the **UML EJB tutorial** EJB project.
    b. Right-click and select **Deploy**.
2. Export the EAR file following these steps:
    a. From the Project Explorer view, highlight the **UML EJB tutorialEAR** Enterprise Application project.
    b. Right-click and select **Export > EAR file**.
    c. Provide a destination directory and a name for the EAR file.
    d. Select **Export source files** if you want the Java source files included in the EAR file.
    e. Select **Finish**.
3. Create the database tables for the test server using the .ddl file. Each database type uses a different process for running .ddl files. For instructions, consult the documentation for the type of database you are using.
4. Test the enterprise application on the test server within the workbench. For information, consult the help topic Testing enterprise beans in the universal test client.
5. Create the database tables for the production server using the .ddl file. Each database type uses a different process for running .ddl files. For instructions, consult the documentation for the type of database you are using.
6. Install the EAR file on a production EJB server using the WebSphere Administrative console. Consult the WebSphere Application Server Information Center for your version of WebSphere.

Finish your tutorial by reviewing the materials in the Summary.

Create EJB components using UML modeling tools summary

Congratulations! You have finished this learning exercise. This tutorial has taught you basic concepts about creating container-managed entity beans using UML class diagrams and mapping EJB fields into relational database definitions.

Completed learning objectives

If you have completed all of the exercises, you should now be able to:

- Create an enterprise application (EAR) project
- Create an EJB project
- Create an EJB client project
- Create a UML class diagram
- Generate relational database mapping files
- Deploy an enterprise application
- Export an EAR file
- Understand the process for testing an enterprise application and putting it into production

More information

If you want to learn more about the topics covered in this tutorial, consider the following Redbooks™:

- EJB 2.0 Development with WebSphere Studio Application Developer, SG24-6819-00
- WebSphere Studio Application Developer Version 5 Programming Guide, SG24-6957-00