# Chapter 10 Developing Full-Scale J2EE Applications

The previous two chapters provided some strategies for extending Web applications with J2EE enterprise-tier technology. This chapter continues the path of those chapters and handles topics such as entity beans, consuming resources, assembling applications from multiple code modules, and verifying the J2EE compliance of applications.

## **Entity Beans**

Beginning with the Enterprise JavaBeans 1.1 specification, entity beans have been a required part of the J2EE platform. Entity beans provide a Java *idiom* (method invocation) for accessing relational functionality in addition to the container benefits provided in the J2EE specification, such as transaction support and security. An entity bean allows persistence to be handled by the container *(container-managed persistence,* or "CMP") or by the developer *(bean-managed persistence,* or "BMP"). The difference between the two is that with CMP beans you need to define how the Java representation is mapped to the relational model (the container handles the code generation required to make this happen), whereas in BMP beans you must provide both the data representation and the implementation that reflects changes to the Java object model in the database. One common approach to bean-managed persistence is to use JDBC in the enterprise bean life-cycle methods.

A developer starting with an existing relational database must map the relational model currently in the database (if the database is in use by other applications, schema changes typically are not possible) to an object model. There are many possible complex mappings between an object model and the relational database. Relational database views are analogous; some of the same issues may be encountered during the mapping process (for example, a view can be created that cannot be updated, normally because of constraint violations).

Container-managed persistence provides some development advantages over bean-managed persistence. CMP beans allow you to work on an abstract schema, which is the combination of fields and relationships, independent of the underlying mapping to the relational database. The underlying mapping is provided by the application server, through the native object to relational mapping functionality. This capability extends to queries, which are written based on the abstract schema in the Enterprise JavaBeans Query Language (EJBQL). The query language is similar to the query capability provided in SQL. This separation allows you to provide an object model and implementation that is separate from the mapping, thereby reducing the coupling. Thus, container-managed beans can be developed faster. Some proponents also claim that their runtime performance is better than that of BMP beans because the optimizations can be done in the server.

Bean-managed persistence requires that you implement the life-cycle methods to interact with the persistent storage mechanism. Typically, you would provide and execute SQL queries in the implementation of the lifecycle methods (ejbStore would perform an update, ejbCreate would perform an insert, and eibLoad would perform a select). The disadvantage of doing JDBC directly in the entity bean is that you do not achieve productivity gains and may not achieve increased performance because of the impedance mismatch between the programming language and a relational database. (Programming languages operate on an object at a time, whereas a relational database can operate on sets of data.) The performance increase may not be realized, because one common mapping is between a row and an object instance. In the entity model, set operations are performed by iterating over a collection of Java objects that cause SQL statements operating on a single row to be invoked. You may be able to incorporate existing tuned queries directly into this model or by using a different data representation to enable set-based operations.

The NetBeans IDE provides both a top-down and a bottom-up approach for creating a related set of CMP entity beans. The top-down approach first creates the entity beans that describe the abstract schema (fields and relationships representing the data model). The Sun Java System Application Server provides a facility for automatically generating the required database tables. Or you can have the database schema created and use the application server mapping facility. The bottom-up approach generates the beans based on an existing database schema (the generation will produce CMP entity beans).

## Creating Entity Beans with the Top-Down Approach

Creating an entity bean from the top down is comprised of the following general steps:

- 1. Add a bean to your EJB module, using the Entity Bean template in the New File wizard.
- 2. Select a primary key class for the bean, using the ejb-jar.xml file's visual editor.
- 3. (For BMP beans) Implement the life-cycle methods.
- 4. (For CMP beans) Add container-managed fields. You can generate such fields by right-clicking the entity bean's "logical" node (under the Enterprise Beans node) in the Projects window and choosing Add | CMP Field or by right-clicking in the bean class in the Source Editor and choosing EJB Methods | AddCMP Field.
- 5. (For CMP beans) Add container-managed relationships. You can do so in the ejb-jar.xml file's visual editor.
- 6. Add finder methods. You can generate such methods by right-clicking a bean class in the Projects window and choosing Add | Finder Method or by right-

clicking the bean class in the Source Editor and choosing EJB Methods | Add Finder Method.

- 7. Add business methods. You can generate such methods by right-clicking a bean class in the Projects window and choosing Add | Business Method or by right-clicking the bean class in the Source Editor and choosing EJB Methods | Add Business Method.
- 8. Set up the data source on the application server.
- 9. (CMP only) If the application server's CMP capability is being used, map the abstract schema specified in the entity bean to a persistence model.

#### Creating an Entity Bean

NetBeans IDE provides a wizard for creating entity beans in an EJB project (see Figure 10-1).

#### Figure 10-1



#### New File wizard with the Entity Bean template selected

The wizard collects the minimum information necessary for creation of either a CMP or BMP entity bean. The wizard generates the necessary deployment descriptor entries along with a local home, local component interface (this provides the ability to perform compile-time checking on the bean class for the component interface methods), business interface , and a bean class.

The classes are named following the BluePrints recommendations. The component interface will initially be empty, and the home interface will contain the required findByPrimaryKey method. The primary key will default to String, which can be changed later using the visual editor (which is accessible by double-clicking the ejb-jar.xml file, under the Configuration Files node in the Projects window). The visual editor for the deployment descriptor is shown in Figure 10-2.

#### Figure 10-2

General	CMP Relation	nships XML	Gene	eral			<b>~</b>
interprise NewEnti	e Beans tyEB						
Gener	al						
Name (ejb	-name):	NewEntityBean	]				
Persistenc	е Туре	Container (CMP)					
Abstract S	ichema Name:	NewEntity					
Primary Ke	ey Field	key	~				
Primary ke	y class:	java,lang.String	r .				
Contract of the Contract of Contract		An owner which have been a second sec					
Reentrant	erise Bean	Implementation a	nd Interfaces				
Enterp	i orise Bean ïelds	Implementation a	ind Interfaces				
Enterp	rise Bean Tields	Implementation a	und Interfaces	Local Se	Remote	Remote	Description
Reentrant Enterp CMP F Field Nam key	ields	Implementation a Type Java.lang.String	Local Ge	Local Se	Remote	Remote	Description
Reentrant     Enterp     CMP F     Field Nam     key     Add	ieids	Type java.lang.String Remove	Local Ge	Local Se	Remote	Remote	Description
Reentrant     Enterp     CMP F     Field Nam     key     Add     CMP F	ii ields ne Edit inder Metl	Implementation a Type java.lang.String Remove hods	Local Ge	Local Se	Remote	Remote	Description
	ields ne Edit Edit	Implementation a Type java.lang.String Nods hods	Local Ge	Local Se	Remote	Remote	Description
Enterp     CMP F     Field Nam     key     Add     CMP F     CMP F     CMP S     CMP S	ields ields ields ields ieldt inder Methelect Methelect invironme	Implementation a Type java.lang.String Remove hods nods nt	Local Ge	Local Se	Remote	Remote	Description
Enterp Enterp CMP F Field Nam Rey Add CMP F CMP F CMP F CMP S Bean E Bean 1	ields ields ields ieldr inder Meth elect Meth Environme Details	Implementation a Type java.lang.String Remove hods nods nt	Local Ge	Local Se	Remote	Remote	Description

Visual editor for a CMP bean deployment descriptor

The deployment descriptor entry includes a display name (using BluePrints recommended naming conventions based on the EJB name). The assembly descriptor also is updated to require transactions for all methods.

#### Selecting a Primary Key Class

Select a primary key class for the bean. You can select a primary key class for the bean in the deployment descriptor (ejb-jar.xml file). A Java class representing the unique identifier (key) for each entity type must be selected. A Java class must represent the entire key. Compound keys and fields defined as primitive types require creation of wrapper classes. The wrapper class generally overrides equals and hashCode. For CMP beans, this class must also provide public fields with names matching the CMP fields composing the key.

#### Implementing Life-Cycle Methods

For BMP beans, you need to implement the life-cycle methods. The required work will also be highlighted using TODO comments in the code generated by the bean template.

Typically, a relational database is used for persistence, so this may require configuration (see Consuming J2EE Resources later in this chapter for more information). The <code>ejbStore</code> method must persist the data model from memory to the persistent storage (SQL update command). The <code>ejbLoad</code> method must retrieve data from the persistent model into the Java data model (SQL <code>select</code> statement). The <code>ejbRemove</code> method must remove data from the persistent model (SQL <code>delete</code> statement). Although not required, an <code>ejbCreate</code> statement must add to persistent <code>storage</code> (SQL <code>create</code> statement). The <code>ejbFindByPrimaryKey</code> method is also required, which should determine whether the key is in persistent storage.

#### Adding Container-Managed Fields

For CMP beans, you need to add container-managed fields. Container-managed fields are the data attributed to this entity. Fields can be added via the EJB Methods menu in the Source Editor, the Add submenu of the contextual menu of the bean's node in the Projects window, or the visual editor for the ejb-jar.xml file. The wizards also provide checkboxes to expose the methods in the local interface.

#### Adding Container-Managed Relationships

(CMP only) To add container-managed relationships, select the CMP Relationships view in the ejb-jar.xml file's editor and then click the Add button to open the Add CMP Relationship dialog box (shown in Figure 10-3). A container-managed relationship represents an association between entity beans. The relationship is managed by the container, which means that adding or removing from one side of the relationship is reflected in the other side.

Description:	vame:				
Entity Bean:	NewEntity	/Bean 💌	Entity Bean:	NewEntityBear	n 💌
Role Name:			Role Name:		
Multiplicity:	💽 One	🔿 Many	Multiplicity:	💿 One 🔹 🔿	Many
Cascade	Delete		Cascade	Delete	
Create C	MR Field		Create C	MR Field	
Create C	MR Field		Create C	MR Field	
Create C Field Name: Field Type:	MR Field		Create C Field Name: Field Type:	MR Field	
Create C Field Name: Field Type: Add To Local	MR Field	~	Create C Field Name: Field Type: Add To Local	MR Field	
Create C Field Name: Field Type: Add To Local	MR Field	~	Create C Field Name: Field Type: Add To Local	MR Field	8

#### Figure 10-3

#### Add CMP Relationship dialog box

The Add CMP Relationship dialog box collects the information necessary to generate the necessary deployment descriptor entries and the abstract get and set methods (if navigable).

In the Add CMP Relationship dialog box, the Entity Bean combo box represents the bean on each side of the relationship. The Cascade Delete checkbox enforces referential integrity by removing the dependent bean if the referenced bean is removed. A container-managed relationship (CMR) field (represented as an abstract getter and setter in the bean class) provides the ability to navigate to the other side of the relationship. These fields can be exposed using the Add to Local Interface checkbox.

#### **Adding Finder Methods**

Finder methods represent the ability to obtain a set of entity beans based on a set of search criteria. The Add Finder method dialog box can be accessed by right-clicking a bean class in the Projects window (and choosing Add | Finder Method) or the Source Editor (and choosing EJB Methods | Add Finder Method).

The EJB specification requires that a findByPrimaryKey method be present. CMP beans specify an EJB Query Language (EJBQL) query to define additional finder methods. The EJBQL query must return instances of the abstract schema for this EJB. The default abstract schema name is the enterprise bean name, so a typical bean query would be SELECT OBJECT (0) FROM MyAbstractSchemaName AS 0 WHERE 0.color = ?1. BMP beans need to return the primary key (or keys) that meet the search criteria.

#### Adding Business Methods

Business methods provide the ability to expose services. Entity beans may add business methods to provide additional logic or to expose data. Adding business methods may be done using the EJB Methods menu from the Source Editor contextual menu or from the Add submenu of the contextual menu for the enterprise bean's node Projects window.

#### Setting up a Data Source on the Application Server

A data source must be set up on the application server. This applies to both BMP and CMP beans. BMP beans may be able to take advantage of the Use Database enterprise resource capability, whereas CMP beans generally must specify a reference to the application server's CMP resource. (The form varies among application servers but typically is similar to what is done to set up a standard JDBC data source.)

#### Mapping a Bean's Schema to a Persistence Model

If the application server's CMP capability is being used, the mapping from the abstract schema specified in the CMP entity bean to a persistence model (most commonly a relational database) must be performed. The Sun Java System Application Server provides the ability to create the tables during deployment (so no mapping is necessary in this case). This can be done by specifying the Create Table At Deploy (and, optionally, Drop Table at Undeploy) properties in the IDE's visual editor for the project's sun-ejb-jar.xml file, as shown in Figure 10-4.

#### Figure 10-4

Visual editor for the sun-ejb-jar.xml file (Sun Configuration panel)

Sun Configuration		
	Name:	apping siniy
	Cmp Resource	
	Jndi Name: jdo/pmf	
	Default Resource Principal Name: Default Resource Principal Password:	
	Create Table At Deploy:	Drop Table At Undeploy:
	Pm In Use:	
	Pm Descriptor Cmp Property S	hema Generator Property
	Message Destination W	'ebservice Description
	Message Destination Name Jndi Name	
	New Edit De	lete

Mapping is done by describing how each CMP field and relationship represents a column in the database. As of the EJB 2.1 specification, this mapping is server specific and, thus, must be performed using the application server editor. When performing the mapping using Sun Java System Application Server 8.1, the first step is to create a database schema file from an existing database. The database schema must be created as a peer to the sun-ejb-jar.xml file (which is located in the src/conf folder and can be accessed from the Configuration Files node in the Projects window). A database schema object provides an XML snapshot of the database metadata. The application server uses this during design-time mapping as a way to access database metadata (without requiring a live database connection). This file is also passed to the application server during deployment for use during code generation.

## \*\*\*Proofreading: In following graf & caption, "Cmp Mapping" is the correct capitalization, per screen shots.\*\*\*

Once the database schema object has been created, each CMP bean must specify the database schema which contains the table to which it is mapping (refer to Figure 10-4). This dialog box is launched by double-clicking the node for the sun-ejb-jar.xml file to open its visual editor, selecting the node for the entity bean in the left pane, selecting the Cmp Mapping tab, and clicking the Advanced Settings button. The visual editor with the Cmp Mapping tab selected is shown in Figure 10-5. The Advanced Settings dialog box is shown in Figure 10-6.

💹 🎒 NewEntityBean [EJB]					
	Name: New	vEntityBean			
	Jndi Name:			Pass By Referen	ice: 🔽
	Is Read Only Bea Refresh Period Is	an:			*
	Commit Option:				~
	Bean Carbe	FIBLLE	Finder Queries	Prefetch D	isabled
	Cmp Map	ping	Tor Security Config	Be	an Pool
	Mappings for I	Bean 'NewEntityBea	n'		
	Primary Table:	:	×		ings
	Primary Table: <unmapped> Field Mappings Field</unmapped>	:: Mapped Colu	mn Type	Fetch	ings

Visual Editor for the sun-ejb-jar.xml file (EJB panel with Cmp Mapping tab selected)

<choose a="" elle="" schema=""></choose>	1.5
Consistency Level	
Level:	
None	×.
Version Column (from Primary Table):	
	~
Secondary Tables	
Secondary Tables:	
	Add
	Remove
	Edit

CMP Mapping Advanced Settings dialog box

Figure 10-6

The final step of mapping CMP and relationship fields from a table can be performed from the Cmp Mapping tab (see Figure 10-7).

Figure	1	0-	-7
--------	---	----	----

SalesTaxCodeTblBean	EJB				
ManufactureTblBean [	Name: SalesRep	TblBean			
ProductCodeTblBean [	Jndi Name:			Pass By Referer	nce:
🞒 SalesRepTblBean [E3B	Is Read Only Bean:				~
Office I biBean [EUB]     Office TypeCode ThiBe.	Refresh Period In Sec	onds:			
	Commit Option:				~
🔄 🎒 OrderTblBean [EJB]					
🛓 🎒 ProductTblBean [EJB]	Mapping Properties:				
😥 🎒 DiscountCodeTblBean					
a SalecDenDataThiPase					
aleskehnargi nipegi i	Bean Cache	E 1B1, 1 Einder	Oueries	Pretetch F	)isabled
CustomerTblBean [EJE	Bean Cache Cmp Mapping Mappings for Bear Primary Table:	EJB1.1 Finder Ior I 'SalesRepTblBean	r Queries	Prefetch E	xisabled ean Pool
् म् अवस्त्र विप्रविवेग खिन् ∯ CustomerTblBean [EJE	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL	EJB1.1 Finder Tor SalesRepTblBean	Queries Security Config	Advanced Set	Disabled ean Pool tings
φ ∰ Jaiesropyota Ubbean	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings:	EJB1.1 Finder Jor 'SalesRepTblBean	Coueries     Security Config	Advanced Set	tings
φ ਦਾ Jaiesropuota IUBean	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings: Field	EJB1.1 Finder Tor 'SalesRepTblBean Mapped Column	Cueries	Advanced Set	tings
φ ਦਾ Jaiesropuota IUBean	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings: Field commissionRate	EJB1.1 Finder Tor 'SalesRepTblBean Mapped Column SALES_REP_TBL	Yupe  Type  PECIMAL(4,2)	Advanced Set	tings
Υ Ξαϊεχτορυστα IUBean	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings: Field commissionRate firstName biscOute	EJB1.1 Finder Ior SalesRepTblBean Mapped Column SALES_REP_TBL SALES_REP_TBL	Cueries Security Config	Prefetch D Be Advanced Set Fetch Default Default	tings
u areskepuata (Dibean B—∯ CustomerTbiBean [EJE	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings: Field commissionRate firstName hireDate lastName	EJB1.1 Finder Ior SalesRepTblBean Mapped Column SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL	Cueries Security Config	Prefetch D Ba Advanced Set Fetch Default Default Default	tings
u pareskepuata (Dibean	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings: Field commissionRabe firstName hireDate lastName lastName	EJB1.1 Finde Ior SalesRepTblBean Mapped Column SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL.L SALES_REP_TBL.L	Cueries Security Config	Prefetch D Ba Advanced Set Default Default Default Default Default	tings
u prostanceµoada (Dibean	Bean Cache Cmp Mapping Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings: Field commissionRate firstName hireDate lastWame lastWame lastWame	EJB1.1 Finder Ior SalesRepTblBean SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL	Cueries Security Config	Fetch D Fetch Default Default Default Default None	tings
u prosported Dibean	Bean Cache Cmp Mappings Mappings for Bear Primary Table: SALES_REP_TBL Field Mappings: Field Mappings: Field commissionRate firstName hireDate lastWrSales officeNum orderTblBean	EJB1.1 Finder Ior 'SalesRepTblBeam Mapped Column SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL SALES_REP_TBL	Cueries Security Config	Fetch D Fetch Default Default Default Default None None	tings

CMP mapping in the sun-ejb-jar.xml visual editor

The Automap All button automates the mapping between tables and fields. Using this feature provides full automation for most mappings, which can be further customized later (or even removed using the Unmap All feature).

## Creating Entity Beans with the Bottom-Up Approach

Starting with an existing relational database and generating entity beans that represent a natural mapping is fully automated in NetBeans IDE 4.1. From a single wizard, you can create CMP entity beans, generate CMP fields with types appropriate for the relational mapping, generate relationship fields (along with the relationship cardinality) based on the foreign key and index constraints, and provide the mapping information to the application server (see the top-down approach for details when performed manually). The wizard will detect joined tables, so there may not be a one-to-one mapping between tables and entity beans from the selected database. The wizard will also generate primary key classes if necessary.

To create a set of entity beans based on an existing database:

- 1. In the Projects window, right-click the node of the EJB module that you want to add the beans to and choose New | CMP Entity Beans From Database.
- 2. Complete the wizard to create the mappings.

The wizard contains two steps (see Figures 10-8 and 10-9). The first step specifies how to obtain database metadata. The package for the generated Java classes is also required. The wizard also allows default generation of finder methods and exposing the fields in the component interface. (Make sure the underlying database process is started.) If you are using the Pointbase database that is included with Sun Java System Application Server 8.1, you can start it from the IDE via the Tools | Pointbase menu item.

#### Figure 10-8

Database Source and Beans Location	
Select the Database Source for generating CMP Er           IDBC Connection         < Select from the list >           Database Schema	ntity Beans
Specify the location for the generated CMP entity Project: ThirdEJBProject	beans
Location:         Source Packages           Package:         org.mydomain.myentitybeans	
Generate finder methods for CMP fields Add CMP and CMR fields to local interface	
	Database Source and Beans Location         Select the Database Source for generating CMP Er         Image: Decompetition         Image: Decompetition         Specify the location for the generated CMP entity         Project:         ThirdEJBProject         Location:         Source Packages         Package:         org.mydomain.myentitybeans         Image: Open-rate finder methods for CMP fields         Image: Add CMP and CMR fields to local interface

Database Source and Beans Location page of the New File wizard for the CMP Entity Beans from Database template

Steps	Select Database Tables		
1. Choose File Type 2. Database Source and Beans	Available Tables:		Selected Tables:
Location 3. Select Database Tables	CUSTOMER_TBL DISCOUNT_CODE_TBL MANUFACTURE_TBL MICRO_MARKETS_TBL		
	OFFICE_TBL OFFICE TYPE CODE TBL	Add >	<u>]</u>
	ORDER_TBL PRODUCT_CODE_TBL	< Remove	
	SALES_REP_DATA_TBL	Add All >>	
	SALES_REP_TBL SALES_TAX_CODE_TBL	<< Remove All	
	Entity Beans will also be gener	rated for all reference	d tables.

Select Database Tables page of the New File wizard for the CMP Entity Beans from Database template

If Generate Finder Methods for CMP Fields checkbox is selected, a single argument finder method, along with the corresponding EJBQL query for selecting the collection of entity beans matching the input parameter, is generated for each CMP field.

Selecting the Add CMR and CMP Fields to the Local Interface checkbox will generate a method in the local interface for each CMP and relationship field. This typically is used in conjunction with the transfer object and transfer-object assembler pattern from a session facade to create domain objects based on entity beans. The transfer objects are standard serializable Java classes that represent the data and can be used outside of J2EE components. The transfer object assembler typically is a Java class that can create transfer objects (from an entity bean, in this case). A session facade typically is used to provide a coarse-grained set of services. A session facade can encapsulate entity bean access and reduce coupling (and increase performance) for clients.

The second step of the wizard selects the set of tables to include in the generation. The wizard will perform transitive closure (based on foreign key constraints) to ensure that relationships are generated. The most common scenario is to select a group of logically related beans to generate CMP beans for a single module.

After you select all the tables you need, the IDE will create a set of CMP beans, one per table selected (and maybe others that are part of the transitive closure of the foreign keys/primary keys of the selected table). The Sun Java System Application Server mapping will be correctly configured. You can still tweak it the way you need from the sun-ejb-jar.xml visual editor, as shown in Figure 10-10.

Example 2011 (a) (b) (c) (c) (c) (c) (c) (c) (c) (c) (c) (c	🗄 NewSessionBean 🗙 🖂 🖘	n-eib-iar x				
Egg Jun Course       Ejb         Name:       ManufactureTbl         Indi Name:       ejb/ManufactureTbl         Is Read Only Bean:						
Name:       ManufactureTbl         Indi Name:       ejb/ManufactureTbl         Is Read Only Bean:	ManufactureTbl [EJB]	Ejb				
Indi Name:       ejb/ManufactureTbl       Pass By Reference:         Is Read Only Bean:       Image: State S		Name: Mar	nufactureTbl			
Is Read Only Bean:         Refresh Period In Seconds:         Commit Option:         Mapping Properties:         Cmp Mapping Ior Security Config Bean Pool Bean Cache EJB1.1 Finder Queries Flush At End Of Mehtod         Mappings for Bean ManufactureTbl'         Primary Table:         MANUFACTURE_TBL         Field Mappings:         Field Mappings:         Field Mappings:         Field ManuFACTURE_TBL.ADDR_LN1         VARCHAR(30) Default         addr.n2         MANUFACTURE_TBL.EMAIL         VARCHAR(30) Default         addr.n2         MANUFACTURE_TBL.MADR_LN2         VARCHAR(30) Default         addr.n2         MANUFACTURE_TBL.FAX         VARCHAR(30) Default         email         MANUFACTURE_TBL.FAX         VARCHAR(30) Default         fax         MANUFACTURE_TBL.FAX         VARCHAR(30) Default         ponae         MANUFACTURE_TBL.MAME         VARCHAR(30) Default         ponae         MANUFACTURE_TBL.MAME         VARCHAR(30) Default         phone         MANUFACTURE_TBL.MAME         VARCHAR(30) Default         phone		Jndi Name: ejb,	/ManufactureTbl		Pass By Reference	x [
Is Read Only Bean:  Refresh Period In Seconds:  Commit Option:  Mapping Properties:  Cmp Mapping Ior Security Config Bean Pool Bean Cache EJB1.1 Finder Queries Flush At End Of Mehtod Mappings for Bean 'Manufacture Tbl'  Primary Table:  MANUFACTURE_TBL  Field Mapped Column Type Fetch addrLn1 MANUFACTURE_TBL.ADDR_LN1 VARCHAR(30) Default addrLn2 MANUFACTURE_TBL.ADDR_LN2 VARCHAR(30) Default city MANUFACTURE_TBL.CITY VARCHAR(25) Default fax MANUFACTURE_TBL.FAX VARCHAR(26) Default fax MANUFACTURE_TBL.FAX VARCHAR(12) Default infrNum MANUFACTURE_TBL.MAX VARCHAR(30) Default fax MANUFACTURE_TBL.MAX VARCHAR(30) Default infrNum MANUFACTURE_TBL.MAX VARCHAR(30					-	
Refresh Period In Seconds:         Commit Option:         Mapping Properties:         Cmp Mapping Ior Security Config Bean Pool Bean Cache EJB1.1 Finder Queries Flush At End Of Mehtod         Mappings for Bean 'ManufactureTbl'         Primary Table:         MANUFACTURE_TBL         Field Mappings:         Field Mapped Column       Type         Field Mappings:         Field Mapped Column       Type         Field MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)         Default       MANUFACTURE_TBL.ADDR_LN2         addrLn2       MANUFACTURE_TBL.CTTY         wANUFACTURE_TBL.FAX       VARCHAR(25)         Default       Fax         MANUFACTURE_TBL.FAX       VARCHAR(12)         Default       Default         fax       MANUFACTURE_TBL.MAX         MANUFACTURE_TBL.MAX       VARCHAR(12)         Default       MANUFACTURE_TBL.MAX         mfrNum       MANUFACTURE_TBL.MAX         mane       MANUFACTURE_TBL.MAX         manufacture_TBL.MER_NUM       INTEGER         name       MANUFACTURE_TBL.NHCNM         mane       MANUFACTURE_TBL.NHCNM         mane       MANUFACTURE_TBL.NHCNME         vaRCHAR(30)       Default         man		Is Read Only Be	an:			1
Commit Option:         Mapping Properties:         Cmp Mapping Tor Security Config Bean Pool Bean Cache EJB1.1 Finder Queries Flush At End Of Mehtod         Mappings for Bean 'Manufacture Tbl'         Primary Table:         MANUFACTURE_TBL         Field       Mapped Column         Type       Fetch         addrLn1       MANUFACTURE_TBL.ADDR_LN1         VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.ADDR_LN2         vARCHAR(30)       Default         ity       MANUFACTURE_TBL.FAX         vARCHAR(40)       Default         fax       MANUFACTURE_TBL.FAX         vARCHAR(12)       Default         marNUFACTURE_TBL.FAX       VARCHAR(12)         Default       MANUFACTURE_TBL.FAX         manufacture_TBL.FAX       VARCHAR(30)         marKina       MANUFACTURE_TBL.FAX         MANUFACTURE_TBL.FAX       VARCHAR(12)         Default       MANUFACTURE_TBL.FAX         marKina       MANUFACTURE_TBL.FAX         MANUFACTURE_TBL.FAX       VARCHAR(30)         Default       Fax         MANUFACTURE_TBL.FAX       VARCHAR(30)         MANUFACTURE_TBL.FAX       VARCHAR(30)         ManufActure_TBL.FAX       VARCH		Refresh Period I	in Seconds:			
Mapping Properties:         Cmp Mapping Ior Security Config Bean Pool Bean Cache EJB1.1 Finder Queries Flush At End Of Mehtod         Mappings for Bean 'ManufactureTbl'         Primary Table:         MANUFACTURE_TBL         Field Mappings:         Field Mappings:         Field Mappings:         Field Mappings:         Field Mappings:         Field Mapping:         Yearchar(30)         Default         addrLn2         MANUFACTURE_TBL.ADDR_LN2         VARCHAR(30)       Default         email       MANUFACTURE_TBL.RMAIL         VARCHAR(40)       Default         fax       MANUFACTURE_TBL.COPMANY         BIT       Default         mfrNum       MANUFACTURE_TBL.NAME         NANUFACTURE_TBL.NAME       VARCHAR(30)         Default       manum         NanuFACTU		Commit Option:				6
Mapping Properties:         Cmp Mapping Ior Security Config Bean Pool Bean Cache EJB1.1 Finder Queries Flush At End Of Mehtod         Mappings for Bean 'ManufactureTbl'         Primary Table:         MANUFACTURE_TBL         Field Mappings:         Field Mapped Column       Type         Addranced Settings         Field Mappings:         Field MANUFACTURE_TBL.ADDR_LN1         VARCHAR(30)         Default         addrLn2         MANUFACTURE_TBL.ADDR_LN2         VARCHAR(30)         Default         city         MANUFACTURE_TBL.CITY         VARCHAR(25)         Default         fax         MANUFACTURE_TBL.FAX         VARCHAR(12)         Default         fax         MANUFACTURE_TBL.MARL_COMPANY         BIT         Default         mfrNum       MANUFACTURE_TBL.NAME         VARCHAR(30)       Default         phone       MANUFACTURE_TBL.NAME         VARCHAR(30)       Default         rep       MANUFACTURE_TBL.REP         vARCHAR(30)       Default         mame       MANUFACTURE_TBL.REP         vARCHAR(30)       Default <td></td> <td></td> <td></td> <td></td> <td></td> <td>19</td>						19
Cmp Mapping       Ior Security Config       Bean Pool       Bean Cache       EJB1.1 Finder Queries       Flush At End Of Mehtod         Mappings for Bean 'ManufactureTbl'       Primary Table:       Advanced Settings         MANUFACTURE_TBL       Image: Comparison of the set of the		Mapping Propert	ties:			
Cmp Mapping       Ior Security Config       Bean Pool       Bean Cache       EJB1.1 Finder Queries       Flush At End Of Mehtod         Mappings for Bean 'ManufactureTbl'       Primary Table:       Advanced Settings         MANUFACTURE_TBL       Image: Company ManufactureTbl'       Advanced Settings         Field       Mapped Column       Type       Fetch         addrLn1       MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)       Default         email       MANUFACTURE_TBL.CTIY       VARCHAR(25)       Default         fax       MANUFACTURE_TBL.EMAIL       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.MAX       VARCHAR(12)       Default         mfrNum       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default       Fetault         mome       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default       Fetault       Fetault         mome       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default       Fetault       Fetault <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
Mappings for Bean 'ManufactureTbl'         Primary Table:         MANUFACTURE_TBL         Field Mappings:         Field Mapped Column       Type         Field Mapped Column       Type         Addrini       MANUFACTURE_TBL.ADDR_LN1         VARCHAR(30)       Default         addrini       MANUFACTURE_TBL.ADDR_LN2         VARCHAR(30)       Default         addrini       MANUFACTURE_TBL.ADDR_LN2         VARCHAR(30)       Default         email       MANUFACTURE_TBL.CITY         VARCHAR(40)       Default         fax       MANUFACTURE_TBL.FMAIL         vARCHAR(12)       Default         icalCompany       MANUFACTURE_TBL.MAR         vARCHAR(30)       Default         mfrNum       MANUFACTURE_TBL.NAME       VARCHAR(30)         phone       MANUFACTURE_TBL.NAME       VARCHAR(30)         phone       MANUFACTURE_TBL.REP       VARCHAR(30)         varchar(12)       Default       varchar(30)         rep       MANUFACTURE_TBL.STATE       CHAR(2)		Cmp Mapping I	or Security Config   Bean Pool   Bean Cach	e EJB1.1 Finder Oueries	Flush At End Of M	lehtod
Primary Table:       Advanced Settings         MANUFACTURE_TBL       Advanced Settings         Field Mappings:       Field         Field Mapped Column       Type         Field MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)         Default       addrLn1         MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)         Default       city         MANUFACTURE_TBL.CITY       VARCHAR(25)         Default       fax         MANUFACTURE_TBL.EMAIL       VARCHAR(12)         Default       fax         MANUFACTURE_TBL.ICCAL_COMPANY       BIT         Default       mfrNum         MANUFACTURE_TBL.NAME       VARCHAR(30)         Default       pone         MANUFACTURE_TBL.NAME       VARCHAR(30)         Default       mane         MANUFACTURE_TBL.NAME       VARCHAR(30)         Default       mane         MANUFACTURE_TBL.REP       VARCHAR(30)         Default       mane         MANUFACTURE_TBL.REP       VARCHAR(30)         Default       mane		-Manninge for	Pean 'ManufactureThl'			
Primary Table:       Advanced Settings         MANUFACTURE_TBL       Advanced Settings         Field Mappings:       Field       Type       Fetch         addrLn1       MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)       Default         city       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(25)       Default         email       MANUFACTURE_TBL.CITY       VARCHAR(40)       Default         fax       MANUFACTURE_TBL.ICAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.IOCAL_COMPANY       BIT       Default         mmm       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		Mappings ion	Dean Manulacture (Di			
MANUFACTURE_TBL       Advanced Settings         Field Mappings:       Field Mapped Column       Type       Fetch         addrLn1       MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.CADDR_LN2       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.CITY       VARCHAR(25)       Default         email       MANUFACTURE_TBL.FAX       VARCHAR(10)       Default         fax       MANUFACTURE_TBL.FAX       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.NOCAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.PHONE       VARCHAR(30)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		Primary Table:				
Field Mappings:         Field Mapped Column       Type       Fetch         addrLn1       MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)       Default         city       MANUFACTURE_TBL.CITY       VARCHAR(25)       Default         email       MANUFACTURE_TBL.FMAIL       VARCHAR(40)       Default         fax       MANUFACTURE_TBL.FAX       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.NOCAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.PHONE       VARCHAR(30)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default						
Field       Mapped Column       Type       Fetch         addrLn1       MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)       Default         city       MANUFACTURE_TBL.CITY       VARCHAR(2S)       Default         email       MANUFACTURE_TBL.FAX       VARCHAR(40)       Default         fax       MANUFACTURE_TBL.FAX       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.OCAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.PHONE       VARCHAR(30)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		MANUFACTUR	E_TBL	×	Advanced Settin	gs
addrLn1       MANUFACTURE_TBL.ADDR_LN1       VARCHAR(30)       Default         addrLn2       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)       Default         city       MANUFACTURE_TBL.CITY       VARCHAR(2S)       Default         email       MANUFACTURE_TBL.EMAIL       VARCHAR(40)       Default         fax       MANUFACTURE_TBL.FAX       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.MARE       VARCHAR(12)       Default         mfrNum       MANUFACTURE_TBL.MARE       VARCHAR(30)       Default         name       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		MANUFACTUR Field Mappings	IE_TBL	×	Advanced Settin	gs
addrLn2       MANUFACTURE_TBL.ADDR_LN2       VARCHAR(30)       Default         city       MANUFACTURE_TBL.CTIY       VARCHAR(25)       Default         email       MANUFACTURE_TBL.EMAIL       VARCHAR(40)       Default         fax       MANUFACTURE_TBL.FAX       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.IOCAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         name       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		Field Mappings	KE_TBL :: ∫ Mapped Column	Туре	Advanced Settin	gs
city     MANUFACTURE_TBL.CITY     VARCHAR(25)     Default       email     MANUFACTURE_TBL.EMAIL     VARCHAR(40)     Default       fax     MANUFACTURE_TBL.FAX     VARCHAR(12)     Default       localCompany     MANUFACTURE_TBL.LOCAL_COMPANY     BIT     Default       mfrNum     MANUFACTURE_TBL.NAME     INTEGER       name     MANUFACTURE_TBL.NAME     VARCHAR(30)     Default       phone     MANUFACTURE_TBL.RP     VARCHAR(30)     Default       rep     MANUFACTURE_TBL.RP     VARCHAR(30)     Default       state     MANUFACTURE_TBL.STATE     CHAR(2)     Default		Field Mappings	E_TBL :: Mapped Column MANUFACTURE_TBL.ADDR_LN1	Type VARCHAR(30)	Advanced Settin Fetch Default	gs
email       MANUFACTURE_TBL.EMAIL       VARCHAR(40)       Default         fax       MANUFACTURE_TBL.FAX       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.LOCAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.NMR_NUM       INTEGER         name       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.PHONE       VARCHAR(12)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		Field Mappings Field Mappings Field addrLn1 addrLn2	E_TBL Wapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2	Type VARCHAR(30) VARCHAR(30)	Advanced Settin Fetch Default Default	gs
fax       MANUFACTURE_TBL.FAX       VARCHAR(12)       Default         localCompany       MANUFACTURE_TBL.LOCAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.MER_NUM       INTEGER         name       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.PHONE       VARCHAR(12)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		Field Mappings Field Mappings Field addrLn1 addrLn2 city	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY	Type VARCHAR(30) VARCHAR(30) VARCHAR(25)	Advanced Settin Fetch Default Default Default	gs
localCompany       MANUFACTURE_TBL.LOCAL_COMPANY       BIT       Default         mfrNum       MANUFACTURE_TBL.MFR_NUM       INTEGER         name       MANUFACTURE_TBL.NAME       VARCHAR(30)       Default         phone       MANUFACTURE_TBL.PHONE       VARCHAR(12)       Default         rep       MANUFACTURE_TBL.REP       VARCHAR(30)       Default         state       MANUFACTURE_TBL.STATE       CHAR(2)       Default		Field Mappings Field Mappings addrLn1 addrLn2 city email	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY MANUFACTURE_TBL.EMAIL	Type VARCHAR(30) VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40)	Advanced Settin Fetch Default Default Default Default	gs
mfrNum     MANUFACTURE_TBL.MFR_NUM     INTEGER       name     MANUFACTURE_TBL.NAME     VARCHAR(30)     Default       phone     MANUFACTURE_TBL.PHONE     VARCHAR(12)     Default       rep     MANUFACTURE_TBL.REP     VARCHAR(30)     Default       state     MANUFACTURE_TBL.STATE     CHAR(2)     Default		MANUFACTUR Field Mappings Field addrLn1 addrLn2 city email fax	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY MANUFACTURE_TBL.EMAIL MANUFACTURE_TBL.FAX	Type VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40) VARCHAR(12)	Advanced Settin Fetch Default Default Default Default Default	gs
name     MANUFACTURE_TBL.NAME     VARCHAR(30)     Default       phone     MANUFACTURE_TBL.PHONE     VARCHAR(12)     Default       rep     MANUFACTURE_TBL.REP     VARCHAR(30)     Default       state     MANUFACTURE_TBL.STATE     CHAR(2)     Default		MANUFACTUR Field Mappings Field addrLn1 addrLn2 city email fax localCompany	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY MANUFACTURE_TBL.FAX MANUFACTURE_TBL.FAX MANUFACTURE_TBL.LOCAL_COMPANY	Type VARCHAR(30) VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40) VARCHAR(12) BIT	Advanced Settin Fetch Default Default Default Default Default Default	gs
phone         MANUFACTURE_TBL.PHONE         VARCHAR(12)         Default           rep         MANUFACTURE_TBL.REP         VARCHAR(30)         Default           state         MANUFACTURE_TBL.STATE         CHAR(2)         Default		MANUFACTUR Field Mappings Field addrLn1 addrLn2 city email fax localCompany mfrNum	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.EMAIL MANUFACTURE_TBL.FMAIL MANUFACTURE_TBL.FAX MANUFACTURE_TBL.IOCAL_COMPANY MANUFACTURE_TBL.INFR_NUM	VARCHAR(30) VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40) VARCHAR(12) BIT INTEGER	Advanced Settin Fetch Default Default Default Default Default Default	gs
rep MANUFACTURE_TBL.REP VARCHAR(30) Default state MANUFACTURE_TBL.STATE CHAR(2) Default v		MANUFACTUR Field Mappings Field addrLn1 addrLn2 city email fax localCompany mfrNum name	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY MANUFACTURE_TBL.EMAIL MANUFACTURE_TBL.FAX MANUFACTURE_TBL.IACAL_COMPANY MANUFACTURE_TBL.MFR_NUM MANUFACTURE_TBL.MFR_NUM	Type VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(25) VARCHAR(40) VARCHAR(12) BIT INTEGER VARCHAR(30)	Advanced Settin	gs
state MANUFACTURE_TBL.STATE CHAR(2) Default		Field Mappings Field Mappings Field addrLn1 addrLn2 city email fax localCompany mfrNum name phone	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY MANUFACTURE_TBL.EMAIL MANUFACTURE_TBL.FAX MANUFACTURE_TBL.INFR_NUM MANUFACTURE_TBL.MARE MANUFACTURE_TBL.MARE MANUFACTURE_TBL.PHONE	Type VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40) VARCHAR(40) VARCHAR(12) BIT INTEGER VARCHAR(30) VARCHAR(12)	Advanced Settin	gs
		MANUFACTUR Field Mappings Field addrLn1 addrLn2 city email fax localCompany mfrNum name phone rep	E_TBL Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY MANUFACTURE_TBL.FAX MANUFACTURE_TBL.FAX MANUFACTURE_TBL.MFR_NUM MANUFACTURE_TBL.MFR_MUM MANUFACTURE_TBL.PHONE MANUFACTURE_TBL.PHONE MANUFACTURE_TBL.PHONE	Type VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40) VARCHAR(12) BIT INTEGER VARCHAR(30) VARCHAR(12) VARCHAR(30)	Advanced Settin	gs
		MANUFACTUR Field Mappings Field addrLn1 addrLn2 city email fax localCompany mfrNum name phone rep state	MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.CITY MANUFACTURE_TBL.FAX MANUFACTURE_TBL.FAX MANUFACTURE_TBL.FAX MANUFACTURE_TBL.NAME MANUFACTURE_TBL.NAME MANUFACTURE_TBL.NAME MANUFACTURE_TBL.NAME MANUFACTURE_TBL.STATE	Type VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40) VARCHAR(40) VARCHAR(12) BIT INTEGER VARCHAR(30) VARCHAR(30) VARCHAR(30) CHAR(2)	Advanced Settin	gs
		MANUFACTUR Field Mappings Field addrLn1 addrLn2 city email fax localCompany mfrNum name phone rep state	Mapped Column MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN1 MANUFACTURE_TBL.ADDR_LN2 MANUFACTURE_TBL.EMAIL MANUFACTURE_TBL.FMAIL MANUFACTURE_TBL.FMAIL MANUFACTURE_TBL.NAME MANUFACTURE_TBL.NAME MANUFACTURE_TBL.NAME MANUFACTURE_TBL.NAME MANUFACTURE_TBL.STATE	Type VARCHAR(30) VARCHAR(30) VARCHAR(25) VARCHAR(40) VARCHAR(40) VARCHAR(12) BIT INTEGER VARCHAR(30) VARCHAR(30) CHAR(2) <u>Automap A</u>	Advanced Settin Fetch Default Unmap A	gs

Figure 10-10 Visual editor for the sun-ejb-jar.xml file with a CMP bean selected

## Assembling J2EE Applications

A J2EE application is a collection of web applications and EJB modules that interact with one another and that can be assembled, deployed, and executed as a unit. The NetBeans IDE takes advantage of the capability to declare dependencies between projects in defining a J2EE application project. Previous chapters cover creation of individual web application and EJB Module projects. A NetBeans J2EE Application project aggregates these individual projects.

You can create a J2EE application by opening the New Project wizard and selecting the Enterprise Application template as shown in Figure 10-11. You can then add individual modules (one IDE project per module) to the J2EE application.

#### Figure 10-11

🛞 New Project		
Steps	Choose Project	
1. Choose Project 2	Categories: General Web Enterprise → Mobile Definition Samples	Projects: Enterprise Application Enterprise Application with Existing Sources SDB Module SDB Module with Existing Sources EDB Module with Existing Ant Script
	Description:	]
	Creates a new enterpris EJB module project and Wet project uses an IDE-gener	e application in a standard project. You can also create an p application project in the enterprise application. A standard ated Ant build script to build and run your projects.
	< Back	Next > Finish Cancel Help

New Project wizard with the Enterprise Application template selected

When creating a NetBeans J2EE Application project, the Name and Location page of the wizard (shown in Figure 10-12) gives you the option of creating one empty EJB Module project and one simple Web Application project.

#### Figure 10-12

## *New Project wizard Name and Location page for the Enterprise Application project template*

You can later add web applicationmodules and EJB modules to a J2EE application project (or remove them from the project) through the Project Properties dialog box of the J2EE application project. To do that, right-click the J2EE application's project and choose Properties. Then, in the Project Properties dialog box, select the Packaging node (see Figure 10-13) and click the Add Project button to add the web application or EJB Module projects.

teps	Name and Location		
Choose Project Name and Location	Project Name: EnterpriseApplica	tion4	
	Project Location: C:\Documents	and Settings\Patrick Keegan	Browse
	Project Folder: Documents and	d Settings\Patrick Keegan\EnterpriseApplication4	
	Server: Sun Java System	Application Server 8.1	
	J2EE Version: J2EE 1.4		
	Create EJB Module:	EnterpriseApplication4-EJBModule	
	Create Web Application Module	EnterpriseApplication4-WebModule	
	Recommendation: Souce Level 1.4	should be used in J2EE 1.4 projects.	
	Set Source Level to 1.4		
	Set as Main Project		

### Figure 10-13

regories.			
@ Sources	Archive File Name: Enterp	iseApplication4.ear	
- O Build L O <mark>Packaging</mark> - O Run	Content to exclude (cappot a	vrlude (lass files):	
	**/*.iava		Add Filter
	**/*.form		Add Filderan
	**/.nbattrs		Permise
	Additional content to include:		
	Archive Item	Location In Archive	Add JAR/Folder
	dist/EnterpriseApplication	14-EJB /	Add Library
	dist/EnterpriseApplication	14-Web /	Had Elbrary
			Add Project
			C. Designed
			Remove

*Project Properties dialog box for an Enterprise Application project with the Packaging panel displayed* 

You will see three or more NetBeans IDE projects in the Projects window when you are working with a J2EE Application (see Figure 10-14): the J2EE application project itself that

exposes the application deployment descriptors (application.xml and possibly sunapplication.xml) and then one NetBeans IDE project per module that is part of the application. The J2EE application project will delegate the build steps for all the submodules and will package the resulting EJB JAR files or Web Archive (WAR) files into a deployable application archive called an EAR (Enterprise Archive) file.

#### Figure 10-14



*Projects window showing an Enterprise Application project that contains an EJB module and a web module (each of which is also an IDE project)* 

The deployment descriptor for a J2EE application is called application.xml (under the Configuration Files node in the Projects window). NetBeans IDE 4.1 does not provide a visual editor for this file, mainly because you are unlikely to need to edit it. The IDE automatically updates this file whenever a module is added to or removed from the application.

A J2EE application project can be cleaned, built, executed, and deployed like any other projects in the IDE. You can also verify the application's J2EE compliance by running the Verify Project command on the project. Right-click the project's node to see the menu of commands.

### Importing Existing J2EE Applications

If you have already started developing J2EE application outside of the NetBeans IDE, it is not too late to switch your development environment to the NetBeans IDE. This process will be very easy for you if your source structure already complies with the J2EE BluePrints conventions. It might be a good time to study these conventions and modify your project to follow these guidelines. Your application will be easier to maintain, new developers joining your project will know immediately where things are, and the integration with the NetBeans IDE and its powerful build system (entirely based on Ant) will be straightforward.

#### Importing BluePrints Conventions-Compatible J2EE Application Projects

If you have started your development without the NetBeans IDE, and you are following the Java BluePrints conventions for your project structure, you can use the Enterprise Application with Existing Sources template in the New Project wizard (shown in Figure 10-15).

V New Project		
Steps	Choose Project	
<ol> <li>Choose Project</li> <li></li> </ol>	Categories: General Web <b>Enterprise</b> Mobile Comparison Samples	Projects: Chterprise Application Enterprise Application with Existing EIB Module EIB Module with Existing Sources EIB Module with Existing Ant Script
	7	
	Description:	
	Imports an existing ent enterprise application must Enterprise Applications (see This witzard also imports con projects use an IDE-gene	terprise application into a standard IDE project. The conform to the 12EE BluePrints Project Conventions fo http://java.sun.com/blueprints/code/projectconventi nponents that are nested in the enterprise application. rated Ant build script to build and run your project
	1	

New Project wizard with Enterprise Application with Existing Sources template selected

To import an existing enterprise application:

- 1. Choose File | New Project.
- 2. In the New Project wizard, select the Enterprise folder, select the Enterprise Application with Existing Sources template, and click Next.
- 3. In the Name and Location page of the wizard, fill in the Location field with a valid top-level directory containing a BluePrints-compliant J2EE application.

This directory should contain subdirectories for each module (web application or EJB Module) that comprises the J2EE application, as well as a src/conf subdirectory containing a valid J2EE application deployment descriptor file called application.xml (and, optionally, one called sun-application.xml). If the Finish button does not get enabled, it means that these conditions are not matched.

teps	Name and Location	
. Choose Project 2. Name and Location	Select the folder that contains all the sources for the Enterprise Application. The sources the BluePrint source structure.	have to foll
	Location: C:\Documents and Settings\Patrick Keegan	Browse
	Specify a name and location for the new project.	
	Project Name: EnterpriseApplication5	
	Project Folder: C:\Documents and Settings\Patrick Keegan\EnterpriseApplication5	Browse
	JZEE Version:         JZEE 1.4           Recommendation:         Souce Level 1.4 should be used in JZEE 1.4 projects.	
	Set Source Level to 1.4  Set as Main Project	
	The Location doesnt contain a Blueprint enterprise application.	

*New Project wizard Name and Location page for the Enterprise Application with Existing Sources template* 

- 4. Fill in the Project Name and Project Folder fields to designate a display name and folder to contain the IDE's settings for the project, including the Ant script. The project folder specified should not already contain a build.xml file, because the IDE will generate one automatically and will not replace an existing one.
- 5. After you click Finish, the J2EE application project is created, as are individual projects for each module in the J2EE application.

#### Importing EJB Modules

You can also import into the IDE stand-alone EJB modules (similar to how you import web application projects) by using the EJB Module with Existing Sources template in the New Project wizard.

To import an EJB module:

- 1. Choose File | New Project.
- 2. In the New Project wizard, select the Enterprise folder, select the EJB Module with Existing Sources template, and click Next.
- 3. In the Name and Location page of the wizard, fill in the Location field with a valid top-level directory containing the EJB module.

Your existing EJB module source does not have to adhere to any particular directory structure. You specify the locations of the configuration files, libraries, and source roots. The main requirement is that the module contain a valid ejb-jar.xml deployment descriptor.

The location directory should not contain folders named dist, nbproject, or build or have an existing build.xml file. Warning messages should appear at the bottom of the wizard in such cases.

#### Figure 10-17

eps	Name and Location	
Choose Project Name and Location	Select the folder that contains all the sources for the EJB module.	
Existing Sources and Libraries	Location: ettings\MyNBProjects\MyExistingEJBModule Browse	
	Specify a name and location for the new project.	
	Project Name: MyExistingEJBModule	
	Project Folder: ettings\MyNBProjects\MyExistingEJBModule Browse	
	Add to Enterprise Application: <a></a>	•
	Server: Sun Java System Application Server 8.1	
	J2EE Version: J2EE 1.4	
	Recommendation: Souce Level 1.4 should be used in J2EE 1.4 projects.	
	Set Source Level to 1.4	
	☑ Set as Main Project	

New Project wizard Name and Location page for the EJB Module with Existing Sources template

- 4. On the Existing Sources and Libraries page of the wizard (shown in Figure 10-18), specify the following properties:
  - [lb] **Configuration Files Folder.** Specifies the location of your deployment descriptors and other configuration files. You must have at least a valid ejb-jar.xml deployment descriptor to complete the wizard.
  - [lb] **Libraries Folder.** Specifies the location of the class libraries that the EJB module depends on. All JAR files in this folder are added to the EJB Module project's classpath and packaged with the module for deployment.

😵 New EJB Module with Exist	ing Sources
Steps	Existing Sources and Libraries
<ol> <li>Choose Project</li> <li>Name and Location</li> <li>Existing Sources and Libraries</li> </ol>	Specify the existing folders containing the configuration files, libraries, source packages, and JUnit test packages.         Configuration Files Folder:       C:\Documents and Settings\MyNBProjects\MyExistingEJBModule\src\conf]         Libraries Folder:       Browse
	Sources Package Folders:
	C:\Documents and Settings\MyNBProjects\MyExistingEJBModule\src\java Add Folder
	Remove
	Test Package Folders:
	Add Folder Remove
	< Back Next > Finish Cancel Help

*New Project wizard Existing Sources and Libraries page for the EJB Module with Existing Sources template* 

The IDE scans the folder you designate in the Libraries Folder field only once when you create the project. After the project is created, adding JAR files to this folder outside the IDE does not add them to the module's classpath. You have to add them manually through the Libraries tab of the module's Project Properties dialog box.

- 5. In the Source Package Folders field, add any source root folders. In the Test Package Folders field, add any test package folders (containing JUnit tests).
- 6. Click Finish. A new NetBeans IDE project is created in the Projects window, and you can start using this EJB Module project like any other IDE project.

#### **NetBeans IDE Tip**

The IDE does not convert deployment descriptors from other application servers to Sun Java System Application Server deployment descriptors. An external tool you can use for this purpose, the Sun Java System migration tool, is available free at http://java.sun.com/j2ee/1.4/download.html#migration.

## Consuming J2EE Resources

One of the goals of deployment descriptors in the J2EE platform is to provide a standard way of describing external facilities that need to be available for successful execution of the J2EE application. The declaration in the deployment descriptor references the expected interface available and the name used to locate this instance. This binding and declaration mechanism allows the late binding of resources. (In J2EE role terms, the deployer can change the actual resource used without changing the source code or the J2EE-mandated deployment descriptors.)

A *resource* is an external entity required by a J2EE application that is referenced using the standard deployment descriptor. A large number of resources can be used, but some of the most common are JDBC (via the javax.sql.DataSource interface), JMS (via the javax.jms.\* interfaces), and enterprise beans (using the interfaces exposed by the bean developer).

Several steps are required to incorporate a resource into an application:

- [lb] Declaration of the resource in the standard deployment descriptor
- [lb] Resource setup in the server-specific deployment descriptor, which may also include setup on the server instance itself
- [lb] Acquisition of the resource using JNDI
- [lb] Using the resource in the programming environment

The NetBeans IDE provides commands to automate the use of resources. Typically you realize that a resource should be used while you are writing Java code. The IDE makes it easy to add a resource as you are coding by providing an Enterprise Resources submenu on the contextual menu in the Source Editor for Java classes contained in enterprise projects (web applications and EJB Modules). See Figure 10-19.



#### **Figure 10-19**

Source Editor with the contextual menu open and the Enterprise Resources submenu selected

The Enterprise Resource menu provides a set of commands that automate the use of JMS, JDBC, and enterprise beans. Subsequent sections describe what happens when the wizards are invoked and how to use the wizards.

All the actions are able to incorporate the J2EE BluePrints Service Locator pattern, which encapsulates and caches the JNDI initial context lookup. This pattern aggregates all boilerplate lookup code and provides the potential for performance enhancements.

The NetBeans IDE provides two different service locator strategies: caching and noncaching. The caching strategy uses the singleton pattern and caches both the initial context and the results from the JNDI lookup. The noncaching locator does not cache the lookup results but instead reuses the initial context. The Service Locator templates are available under the Enterprise node in the New File wizard, as shown in Figure 10-20.

Steps	Choose File Type	
1. Choose File Type	Project: 🔇 MyExistingEJBModule	
	Categories:	File Types:
	Dava Classes     Java Classes     JavaBeans Objects     Junit     Databases     Sun Resources     XML     Ant Build Scripts     Other	<ul> <li>Session Bean</li> <li>Entity Bean</li> <li>CMP Entity Beans from Database</li> <li>Message-Driven Bean</li> <li>Service Locator</li> <li>Caching Service Locator</li> </ul>
	Description:	
V	Creates an instance of the Service Loc JNDI lookup code into a single class, re locator pattern stores instances that a pattern is typically used in the web tier	ator JZEE design pattern. A service locator encapsulates the iducing the proliferation of lookup code. The caching service re already resolved, which reduces subsequent lookups. Thi
	4	

#### Figure 10-20

New File wizard with the Caching Service Locator template selected

The Service Locator template generates a Java source file that can be further customized. The service locator last used for the project is stored with the project settings, allowing the last locator to be reused easily. The service locator naming conventions provided in the templates are incorporated into the enterprise resource commands; therefore, changing the name of the public lookup methods after generation may require manual intervention following the use of the enterprise resource commands.

The caching service locator strategy is most useful from a web module where resource definitions are shared across the entire module. The singleton pattern can be used effectively in this scenario to reuse lookup instances. In an EJB module, resource declarations are scoped to a single enterprise bean; thus, the singleton pattern is not applicable. An EJB module will commonly use a noncaching service locator.

#### **Using Enterprise Beans**

If you want to use an enterprise bean, you must go through the following steps:

- 1. Declare an ejb-ref entry or an ejb-local-ref entry (depending on whether the enterprise bean exposes local or remote interfaces) in the deployment descriptor. The deployment descriptor entry will specify the home and component interface, a name used in the JNDI lookup to specify the instance, the type of enterprise bean (either session or entity), and a reference to the actual enterprise bean.
- 2. Perform a JNDI lookup to obtain an instance of the home interface.
- 3. Use a create or finder method on the home interface to obtain an instance of the component interface.

The Call Enterprise Bean command (which you can access by right-clicking the Source Editor and choosing Enterprise Resources | Call Enterprise Bean) automates this process. The dialog box shown in Figure 10-21 provides a way to select the enterprise bean to invoke. This dialog further enables you to specify the enterprise bean, the desired lookup strategy, and whether checked exceptions should be converted to runtime exceptions.

#### Figure 10-21

🕄 Call Enterprise I	3ean	
Choose the enterprise b	ean you want to call.	
📮 🔌 ThirdEJBProje	:t	~
Customer	THEB	
- 🗿 Discount C	odeTblEB	
- 🔐 Manufactu	ureTblEB	
MicroMark	etsTblEB	
NewEntity	EB	
OfficeTblE	В	
OfficeTyp	eCodeTblEB	
OrderTblE	В	
ProductCo	deTblEB	~
Service Locator Strate     Generate Inline L     Existing Class	igy ookup Code	
Convert Checked E:	ceptions to RuntimeException	
Reference Name:	ejb/DiscountCodeTblBean	
Referenced Interface:	⊙ Local ◯ Remote	
	OK Cancel	

#### Dialog box for the Call Enterprise Bean command

Using the Call Enterprise Bean command in an EJB Module project or a Web project will result in the following:

- [lb] Generation of an ejb-ref or ejb-local-ref in the deployment descriptor. If this is done from an EJB module, this feature can be invoked directly only from the bean class. This feature requires that the referenced enterprise bean is in the same deployment unit (either the same module or the same J2EE application) as required by the semantics of the ejb-link (the reference is based on ModuleName#EjbName syntax and requires the reference to be collocated).
- [lb] Generation of the lookup code. If a service locator is being used, the generated code delegates the lookup to the locator; otherwise, lookup code is generated. The generated code uses the JDK logging framework to log the exception. The

lookup code returns the home interface unless a stateless session bean is referenced, in which case an instance of the component interface is returned.

[lb] A project dependency is established between the current project and the referenced project as the interfaces from the enterprise beans need to be used during compilation. The interfaces will not be included in the archive of the referencing project but will instead assume that the server supports delegation to the application class loader (for references outside a module). If this is not true, the Java Extension mechanism can be used to add a classpath reference to the EJB module.

#### Using a Database

The Use Database command automates the process of incorporating a relational database into an application. If you intend to use a database, you must go through the following steps:

- 1. Declare a resource-ref in the deployment descriptor. The deployment descriptor entry must specify the resource-type javax.sql.DataSource, the ability to share the resource, and the authorization mechanism (typically, the authorization is done by the container).
- 2. Add code to perform a JNDI lookup to obtain an instance of DataSource.
- 3. Use server specific mechanisms to configure the application server to use the JDBC resource. This typically involves adding the JDBC driver to the server classpath, as well as setting up a connection pool for the JDBC connection describing how to connect (including authentication) to the database.

The Enterprise Resources | Use Database command (available by right-clicking a file in the Source Editor) automates this process. When you choose this command, the Choose Database dialog box (shown in Figure 10-22) appears and prompts you to select the database to use in either a web module or an EJB module. In the dialog box, specify a JNDI name (used as the resource reference name), the connection to use, and whether a service locator should be used or inline code should be generated (lookup code is generated in the current class). The list of connections is the same as what is provided in the Runtime window. The Runtime window provides the ability to edit and query the target database.

#### Figure 10-22

S Choose	Database	2	×	
JNDI Name:	myDatabase			
Connection:	jdbc:pointbase://localhost:9092/sample [pbpublic on PB			
( )	Add Driver	Add Connection	٦	
Service Lou Gener Existin	ator Strategy ate Inline Lookup Code ng Class	Cancel		

#### Choose Database dialog box

After you fill in the Choose Database dialog box, the following occurs:

- 1. A resource reference is added to the deployment descriptor using the form jdbc/JNDIName. Existing resource references are reused if possible (matching is done using the description, which initially is populated via the connection string).
- 2. Code is generated to automate the JNDI lookup, using the specified lookup strategy to obtain the data source.
- 3. If the Add Driver and Add Connection buttons are used, a connection node (and also possibly a driver node) is added to the Runtime window.
- 4. Assuming that you are using the Sun Java System Application Server, the necessary steps to enable successful deployment and execution are performed. (Technically, it's the IDE's Sun Java System Application Server integration plug-in module that provides this support. By the time you read this, there might be IDE plug-in modules to provide equivalent features for different servers.) This might involve creating a connection pool and adding the JDBC driver to the classpath of the application server. The connection pool setup can be configured later in the Server Resources node or via the administrative capability of the server. The Server Resources node enables you to provide version control for and deploy server resources that are required to successfully execute an application. The resources provide all the necessary information to configure the setup. This may include usernames and passwords, so editing these files may be necessary when sharing the application with other team members.

#### Sending a JMS Message

The Send JMS Message command automates the process of sending a message to a queue or topic. The J2EE 1.4 specification defines the concept of a message destination, which provides a way to specify logical destinations for messages. A message destination allows the decoupling of the message consumer and message producer. The most common scenario is to have a message driven bean (MDB) linked to the message destination. However, the message destination mechanism allows only the logical message destination to be used by the module sending the message. Although this concept defines a generic linking mechanism, additional information, such as the type of destination (J2EE 1.4 allows other messaging systems to be used), and the message format must be exchanged.

If you want to use JMS, you need to do the following:

- 1. Add a resource reference to declare a ConnectionFactory. The connection factory is used to create JMS queue or topic connections.
- 2. Add a message destination reference that declares the message destination type (which must match the type declared for the message destination), the use (the sender will produce messages), and the destination (in the form of a link). The link will be in the format moduleName#destinationName and must be included in the same application.
- 3. Add code to send the message. The code to send a message must do a JNDI lookup to obtain the connection factory, use the connection factory to create a session, use the session to create a message producer, use the session to create a message, send the message, and close the session and connection.

The Send JMS Message command automates this process. To use this command, right-click in the Source Editor and choose Enterprise Resources | Send JMS Message. The Select Message Destination dialog box (shown in Figure 10-23) appears and provides a way to select a message destination and to specify whether a service locator should be used or inline code should be

generated (lookup code is generated in the current class). Only projects that can supply a destination are shown in the list of projects (web and EJB Modules).

Select /	Message Destination	
The selected	project must be in the same dep	oloyment unit
Project:	MyExistingEJBModule	~
Destination:		~
Service Lo	cator Strategy ate Inline Lookup Code	
O Existin	ng Class	
	OK Cancel	

#### **Figure 10-23** Select Message Destination dialog box

After you complete the Select Message Destination dialog box, the following occurs:

- 1. A resource reference is added to the deployment descriptor, specifying the intention to use a ConnectionFactory. The IDE's Sun Java System Application Server integration plug-in module shares the connection factory with the message consumer. This can be changed by creating a new ConnectionFactory, if desired.
- 2. A message destination reference is created and linked to the selected destination. The message type is discovered by determining the message type of the consumer linked to the destination.
- 3. A private method providing the code necessary to send a single JMS message (named sendJMSDestinationName) is created. This method handles connection, session, and producer creation and destruction. This method accepts a context parameter that allows data to be passed via parameters instead of instance variables, although instance variables can be used as well.
- 4. A second private method (createJMSMessageForDestinationName) is added to create the message (this is the template pattern) from the supplied session and potentially the context parameter.

## J2EE and Security Management

The J2EE platform offers a rich environment for securing web applications, web services, and enterprise beans in a declarative manner by working with application resources and user roles. The two concepts are defined as follows:

- [lb] Resources are visible or callable features of the applications. For EJB modules, resources are public EJB methods declared on home or remote interfaces. For web modules, resources are URL patterns that are mapped to JavaServer Pages (JSP) files, servlet methods, and other components.
- [lb] Roles define access privileges and can be associated with one or more users.

J2EE applications are secured by mapping resource to roles. When a resource is called, the caller must supply a role name that is authorized to access the resource. If the caller cannot supply an

authorized role, the call is rejected. In J2EE applications, the application server verifies the caller's role before allowing the caller to execute the resource.

The authorized combinations of roles and resources are declared in deployment descriptors. The application server reads them from the deployment descriptors and applies them. This process is known as *declarative security*.

The necessary tasks that you must perform to secure a J2EE application are:

- [lb] Declare the different roles.
- [lb] Specify which roles are permitted to access the resources.

This section describes the different steps to follow to secure a simple web application with the NetBeans IDE. For a complete tutorial on J2EE and security, you can refer to the J2EE tutorialathttp://java.sun.com/j2ee/1.4/docs/tutorial/doc/Security2.html.

#### Simple Declarative Security

If you want to secure the access for the web pages exposed within a J2EE application, you need to declare <security-constraint>, <security-role>, and <login-config> elements in the web.xml deployment descriptor (which you can find in the Project window by expanding the web project's Configuration Files node). The visual web.xml editor does not expose those elements in NetBeans IDE 4.1, so you need to switch to the XML view and add the following elements:

```
<security-constraint>
 <web-resource-collection>
   <web-resource-name>basic security test</web-resource-name>
   <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>staffmember</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>basic-file</realm-name>
</login-config>
<security-role>
  <role-name>staffmember</role-name>
</security-role>
```

These settings protect the access of all the web pages (see the <url-pattern> element), using the BASIC login configuration. The authorized logical user is called staffmember.

#### **Registering Users for an Application Server Instance**

To add authorized users to the Sun Java System Application Server, follow these steps:

1. Make sure the server instance is up and running by opening the Runtime window, right-clicking the server instance's node, choosing Start/Stop Server (as shown in Figure 10-24), and then starting the server if it is stopped.

#### Figure 10-24



Starting the application server from the IDE's Runtime window

- 2. Right-click the server instance's node and choose View Admin Console. The login page for the Admin Console appears in a web browser.
- 3. Log into the application server's Admin Console -, and enter the username and password of a user in the admin-realm who belongs to the asadmin group. The name and password you entered when installing the server will work. The NetBeans IDE/Sun Java System Application Server bundle uses these default values: admin for the username and adminadmin for the password.



Sun Java System Application Server Admin Console after having created a user account

- 4. In the Admin Console tree, expand the Configuration | Security | Realms node and select the file realm to add users you want to enable to access applications running in this realm.
- 5. Click the Manage Users button.
- 6. Click New to add a new user to the realm. In this case, use the username ludo and the password ludo as well. You can also enter a group to which the user belongs, but leave that field blank for this example.
- 7. Click OK to add this user to the list of users in the realm. Figure 10-25 shows the state of the console after entering the user ludo to the realm.8. Click Logout when you have completed this task.

Now that you have registered a username for this application server instance, you can map the J2EE logical security role called staffmember to this physical user called ludo. To do that, use the IDE's visual editor for the sun-web.xml file:

1. Open the sun-web.xml file from the Projects window by expanding the project's Configuration Files node and double-clicking the sun-web.xml node.

- 2. In the visual editor, expand the Sun Web Application node. If you have declared the <security-constraint>, <security-role>, and <login-config> elements as described above in the Simple Declarative Security section, the staffmember node should appear.
- 3. Select the staffmember node, click the Master Roles tab in the right pane of the visual editor, click Add Principal, and type ludo in the New Principal Name dialog box. (Figure 10-26 shows what the visual editor should look like after you have completed this step.)
- 4. Click the Role Assignments tab, select ludo in the Principal Master List and click the Add button to assign that user to the staffmember role. (Figure 10-26 shows what the visual editor should look like after you have completed this step.)

Figure 10-26

Sun Web Application	Security Role Mapping	(
Startmenuer Locurity Kole Mapt	Master List of Principals	
	ludo	Add Principal
		Edit Principal
		Remove Principal(s)
	We describe a figure of	
	Master List of Groups	Add Group
		Edit Group
		Remove Group(s)
	Role Assignments Master Roles	

Visual editor for the sun-web.xml file, Master Roles tab, after having added the principal ludo

#### Figure 10-27

🕃 sun-web.xml * 🗙 🐘 web.xml 🗴			
Sun Web Application	Security Role M	apping	Ø
	Security Role Name : Principal Master List Iudo	staffmember	Principals Assigned to this Role
		Add >	)
	Group Master List		Groups Assigned to this Role
		Add >	
		< Reliuve	
	Role Assignments	Master Roles	

Visual editor for the sun-web.xml file, Role Assignments tab, after having mapped the staffmember role to the user ludo

Now when you run this web application, you are prompted for a username and password when you first try to access the application's welcome (as shown in Figure 10-28). Enter ludo as the username and ludo as the password, and you should be able to access the requested web page.

#### Figure 10-28

Promp	1
?	Enter username and password for "basic-file" at http://localhost:1228 User Name:
	ludo
	Password:
	****
	Use Password Manager to remember these values.
	OK Cancel

Prompt for username and password to access a web application that has security constraints set up

This section is only an introduction to J2EE security settings. Make sure you read the J2EE tutorial document, which covers more advanced security concepts for J2EE applications, such as web-services message protection for service endpoints (WSS in the SOAP layer is the use of XML Encryption and XML Digital Signatures to secure SOAP messages). You can find a complete tutorial at the following URL: http://docs.sun.com/source/819-0079/dgsecure.html#wp14462.

Remember, once you are ready to use more advanced security settings, you can take advantage of the IDE's visual configuration editors (for sun-ejb-jar.xml and sun-web.xml) to edit these security settings.

## Understanding the J2EE Application Server Runtime Environment

The Sun Java System Application Server integrates nicely with the NetBeans IDE, via both well-published interfaces (JSR 88) for configuration and deployment and NetBeans IDE extensions called the J2EE Server Integration APIs (see http://j2eeserver.netbeans.org). Although the most complete runtime management tool for the application server is the Admin Console (a management tool in the form of a web page shown in Figure 10-25), some developer-oriented management features have been exposed from the IDE's Runtime window under the Servers node. From there, for each registered server instance, you can view the server log file (for local servers), launch the Admin Console tool, explore the list of deployed J2EE applications or J2EE resources (and undeploy them), and more.

#### **NetBeans IDE Tip**

If you want to explore the entire content of an application server domain or to see the configuration files, the log files, or the repository of the deployed applications, you can add this domain directory in the IDE's Favorites window.

By default, the Favorites window is not visible within the IDE, but you can display it in the explorer area of the IDE by choosing Windows | Favorites. Then you can right-click the Favorites root node, choose Add to Favorites, and navigate to the application server's domain directory. Once you have added this directory in the Favorites window, you can view those files in the IDE's Source Editor and take advantage of other IDE features, such as the Validate XML command, which is useful for validating things such as the config/domain.xml file.

#### Server Log

The Sun Java System Application Server instance container process uses a log file to track system and user messages ranging from severe errors to informational messages describing the current state of the server. The server log file is an important source of information that a J2EE developer needs to use actively during development or deployment and execution of J2EE applications. There is one server log file per instance, usually located under the domain directory in the logs/server.log file. The NetBeans IDE can display the content of the most recent lines of the server log via the server instance node popup menu called View Server Log. This menu is active only for local server instances, because the IDE needs access to the file.

#### Server Management

For a registered application server instance, whether the server is local (on the same machine that the IDE is running on) or remote, the IDE offers extensive administration capabilities via the Runtime window tree view (shown in Figure 10-29).



Runtime window with the node for a Sun Java System Application Server instance selected

Once the server is running, you can get read/write access to most of the developer-oriented administration parameters, such as:

- [lb] The list of J2EE deployed applications, from where you can introspect their properties and undeploy them
- [lb] The list of registered resources and their properties (JDBC resources, connection pools, Persistence Manager resources, JMS resources, JavaMail resources, JDNI names, and Connector and JVM settings)

Each of these management artifacts is represented as a node. The corresponding commands (Delete, Undeploy, Refresh, and so on) are available as popup menu items(when you right-click the node), and corresponding properties are available via the Properties popup menu item. For other management artifacts, you can always access them by opening the application server's Admin Console (right-click the server's node in the Runtime window and choose View Admin Console). This command opens a web browser that then displays the URL for the administration console.

#### JVM Options

The JVM parameters of a server instance are among the most important server administration parameters from a developer standpoint. You can access these parameters by right-clicking the server's JVMs node in the Runtime window and choosing Properties. The Properties dialog box for the JVMs node is shown in Figure 10-30.

🛞 JVMs - Properties	
Properties	
ClasspathSuffix	\${com.sun.aas.installRoot}/pointbase/lib/ 🛄
DebugEnabled	false 😪
NativeLibraryPathPrefix	
JavacOptions	-g 🛄
ClasspathPrefix	null
BytecodePreprocessors	
DebugOptions	-Xdebug -Xrunjdwp:transport=dt_shmem,
EnvClasspathIgnored	true 🛛
JavaHome	\${com.sun.aas.javaRoot} 📃
ServerClasspath	\${com.sun.aas.installRoot}/lib/install/appli
RMICOptions	-iiop -poa -alwaysgenerate -keepgenerat 🛄
JVMOptions	-client -Djava.endorsed.dirs=\${com.sun [
NativeLibraryPathSuffix	
JVMs	0
	Close

Property sheet for the JVMs node for an application server in the Runtime window

Use the server-classpath entry (shown as ServerProperty in the property sheet) when you want to add shared libraries or JAR files that would be used by all the J2EE applications deployed to this server instance, as well as the necessary JDBC driver JAR files to access databases. Use the jvm-options parameter (shown as JVMOptions in the property sheet) to define Java options like HTTP proxies or options that would be relevant to the deployed J2EE applications. You can modify the java-home entry (shown as JavaHome in the property sheet) to change the location of the JDK used by the server instance.

Most of these settings require a server instance restart to take effect. The NetBeans IDE can figure out when a local instance has to restart, and at the next J2EE project deployment, execution, or debugging session, the server is restarted automatically. For remote servers, you have to log into the remote machine to perform a server restart, as there is no command (CLI or Admin Console based) to restart a remote server in Sun Java System Application Server 8.1.

#### **JDBC Drivers**

Using the Sun Resources template category in the New File wizard (shown in Figure 10-31), you can define J2EE resources such as:

- [lb] JDBC connection pools
- [lb] JDBC resources
- [lb] JMS resources
- [lb] JavaMail resources
- [lb] Persistence resources



New File wizard with the JDBC Connection Pool template selected

The wizard for the JDBC Connection Pool template (shown in Figure 10-32) is of particular help in improving your productivity. It enables you to create resources, either from live database connections registered within the NetBeans IDE or from a predefined list of well-known JDBC drivers— including all the Sun DataDirect drivers for DB2, Oracle, Microsoft SQL Server, and Sybase, as well as drivers for the Pointbase and Apache Derby databases. Note that in the Sun Java System Application Server 8.1 Platform Edition, only the Pointbase driver is provided. The Sun Datadirect drivers are included with the Standard Edition and Enterprise Edition of Sun Java System Application Server 8.1.



Choose Database Connection page for the JDBC Connection Pool template in the New File wizard

These server-specific resources are created under the setup directory, shown in the Projects window under the Server Resources node of the project. These server resources are then either automatically deployed to the server whenever the project is executed or manually registered, using the Register menu item, as shown in Figure 10-33.

#### Figure 10-33



#### Registering a database connection in a project

Double-clicking a server resource node opens the property sheet for this resource,, where you can modify all the necessary properties before doing a registration.

#### Figure 10-34

Property sheet for a JDBC connection pool

Properties		
Connection Validation Method	auto-commit	~
Description	null	
DataSource Classname	com.pointbase.jdbc.jdbcDataSource	
Fail All Connections	false	V
Connection Idle Timeout (secs)	300	
Is Connection Validation Required	false	~
Is Isolation Level Guaranteed	true	~
Maximum Pool Size	32	
Maximum Wait Time (msecs)	60000	
JDBC Connection Pool Name	connectionPool	C
Pool Resize Quantity	2	
Resource Type	javax.sql.DataSource	~
Steady Pool Size	8	
Transaction Isolation Level	JDBC Driver Default	~
Validation Table Name	TAB_NAME	
Properties	3 Extra Properties	
connectionPool Properties of JDBC Connection Pool Rest	jurce	C

#### **Server Properties**

The server's Properties dialog box (shown in Figure 10-35), accessible via the Properties menu item of a server node, allows the editing of the admin username or password, as well as the setting for enabling the IDE's HTTP Monitor. This HTTP Monitor can work only for local application servers.

Server Name:	Sun Java System Application Server 8.1 [running]	
Server Type:	Sun Java System Application Server 8.1	
Connection	Classes Sources Javadocs	
Location:	localhost:4848	
Username:	admin	
Password:		
Domains folde	er: E:\Sun\AppServer81\domains\	
Instance Nam	ne: domain1	
Enable H	ITTP Monitor	

Properties dialog box for the Sun Java System Application Server

#### Using JavaServer Faces Technology in a Web Application

Sun Java System Application Server 8.1 bundles all the latest JavaServer Faces (JSF) APIs and the JSF implementation out of the box, and these libraries are recognized by the NetBeans IDE. The JSF config DTDs are also registered in the IDE, meaning that XML code completion and validation are available in the IDE for the JSFconfig 1.0 and 1.1 XML files.

The JSF libraries are automatically in the web application project classpath when the project targets the Sun Application Server, so that all the classes are available to import and to use from your web application.

Furthermore, the Java BluePrints Solution Catalog is accessible directly through the IDE's Help | BluePrints Solutions Catalog menu item and contains some web tier solutions that incorporate JSF technology, which are installable as IDE projects with one click. Install a solution, run the project, study it, debug it, modify it, and use it as a starting point for your JSF project. See Chapter 7 for more details.

#### Working with Ant

The NetBeans IDE's Ant-based project system (introduced in Version 4.0 and extended in Version 4.1) is perhaps the biggest thing that sets NetBeans apart from other IDEs. All projects created within the NetBeans IDE rely on the Ant tool for building. This is also true for the J2EE project types like web applications, EJB modules, and J2EE applications. Therefore, these projects can be built either from the IDE or from outside the IDE using Ant 1.6.2.

The IDE-generated build script also includes targets for executing, debugging, and deploying the application. In NetBeans IDE 4.1, it is not possible to use these targets outside the IDE, because they would require an operational runtime environment. This limitation should go away in NetBeans IDE 4.2.

Another NetBeans IDE advantage is that all of the Sun Application Server 8.1 Ant optional tasks are registered in the IDE. These tasks include

- [lb] sun-appserv-deploy
- [lb] sun-appserv-undeploy
- [lb] sun-appserv-instance
- [lb] sun-appserv-component
- [lb] sun-appserv-admin
- [lb] sun-appserv-input
- [lb] sun-appserv-update

So if you have existing Ant scripts that are using these Ant task extensions, you can run these build.xml files directly from the IDE itself as though you were invoking the Sun Application Server tool asant (which itself is a wrapper around Ant to declare these extensions).

For example, if you download all the J2EE samples for the Sun Application Server, all the samples can be built from the IDE using the sample build.xml files provided with the sample. You could make the samples visible in the IDE by choosing Window | Favorites, right-clicking the Favorites node, choosing Add to Favorites, and navigating to the directory to display. Once the samples are visible in the Favorites window, you can navigate to a sample's build.xml file, right-click it, and choose an Ant target to run.

For more information regarding these tasks, refer to the application server documentation at http://docs.sun.com.

#### Ensuring J2EE Compliance

The J2EE platform extends the many benefits of the Java language to the enterprise, such as portability, by providing a set of specifications for application servers and an extensive set of compatibility tests. The J2EE platform allows the ability to run enterprise applications on a variety of J2EE application servers.

The IDE provides a rich environment for developing J2EE applications. But one of your most crucial objectives as a J2EE developer is ensuring that your application is J2EE compliant. By being J2EE-compliant, the application will deploy and run in any J2EE vendor's application server as long as that vendor's application server is compliant with the J2EE specification. To help you ensure that your J2EE application is J2EE compliant, the IDE provides a built-in J2EE verifier. The NetBeans IDE J2EE Verifier is a tool intended to help you test your web application or enterprise application for the correct use of J2EE APIs. In addition, it confirms portability across J2EE-compatible application servers, which helps you avoid inadvertently writing nonportable code.

When the Verifier is invoked in NetBeans IDE, it runs a series of tests against a selected NetBeans IDE project to ensure that the application is J2EE compliant. The output from running the Verifier is displayed in the NetBeans IDE Output window in table form and contains the following information:

- [lb] Test Status—pass, fail, or warning
- [lb] Test Description—a description of the compatibility test that was run
- [lb] Test Result Description—a detailed description of the test result

You can also filter the Verifier output. You can view all compatibility test results, compatibility failures only, or compatibility failures and warnings only.

After you finish developing a web application or enterprise application and before you deploy it, you should run the J2EE Verifier against your project to ensure that you are not using any vendor-specific proprietary extensions to the J2EE platform. Using a proprietary extension will make it difficult to port to other J2EE application servers.

To launch the Verifier against your web application or enterprise application project, you must have your project open in the NetBeans IDE. From the Projects window, right-click the main node of the project you want to verify and choose Verify Project. Figure 10-36 shows the context-sensitive menu that is displayed when you right-click a web application or enterprise application project.

**Figure 10-36** 



From the Projects window, running the Verify Project command for a J2EE application

Once the Verify Project command is chosen, the Verifier window (shown in Figure 10-37) appears at the bottom of the IDE and starts displaying results of the verification.

#### Figure 10-37

Status	Test Description	Result	
Pass	Session bean home interface create For [ MyExistingEJBModule#My		1
Pass	Session bean home interface create.	For [ MyExistingEJBModule#My	192
Pass	Session bean home interface create.	For [ MyExistingEJBModule#My	
Pass	Session bean class declares all field	For [ MyExistingEJBModule#My	1
Pass	Tests the ejb-name that is defined i	. For [ MyExistingEJBModule#My	
Pass	jms-max-messages-load specifies th.	For [ MyExistingEJBModule#My	1
Not Applicable	Enterprise bean business method is .	For [ MyExistingEJBModule#My	
Not Applicable	Enterprise bean business method is .	For [ MyExistingEJBModule#My	~
Detail:			

Output from the Verify Project command

The results consist of all of the compatibility tests performed against your application. The Verifier lists test status, test description, and a detailed test result. The default for the Verifier results is to display all compatibility tests that fail first, followed by those that have a warning result and then those that have passed. When a row or test is selected in the table, a detailed description of the test result is displayed. Also, the Verifier test results display can be filtered by selecting one of the following radio buttons:

- [lb] All Results—displays all of the compatibility test results
- [lb] Failures Only—displays only the compatibility test failures
- [lb] Failures and Warnings Only—displays both compatibility test failures and warnings

You should fix all compatibility test failures before deploying the application. As you make corrections to your web or enterprise application, you can rerun the Verifier by again right-clicking the project's node in the Projects window and choosing Verify.

#### Refactoring Enterprise Beans

Software applications evolve over time and require updates, bug fixes, feature enhancements, and so on. To support this type of development activity, the NetBeans IDE refactoring features extend to J2EE applications.

Enterprise beans are among the types of code most likely to need refactoring during an enterprise software application's lifetime. However, refactoring enterprise beans can be very time consuming as a result of the complexity of changes that may be involved with the refactoring, due to the number of classes and deployment descriptors that may be affected by a refactoring.

For example, a business method rename may require changes not only in the EJB class you are editing, but also in a local interface, remote interface, calling classes (such as a Servlet class or a remote client class), and deployment descriptors. The NetBeans IDE simplifies this task not only by applying the refactoring changes to the Java source code, but also by making those more

difficult changes in deployment descriptors. The NetBeans IDE's refactoring features greatly simplify this task.

Following are some of the things you can do when refactoring your J2EE source code.

- [lb] Rename an enterprise bean
- [lb] Rename a field or method of an enterprise bean
- [lb] Change method parameters (including parameter names, parameter types, and method visibility), add method parameters, and remove method parameters
- [lb] Encapsulate fields by changing their visibility, adding getter and setter accessors, and determining whether to use accessors even when the field is accessible

The NetBeans IDE allows you to review your refactoring changes before committing them. Likewise, the IDE allows you to undo your J2EE refactoring changes after having committed them and even to revert refactoring changes to deployment descriptors.

#### NetBeans IDE Tip

If you want to undo a refactoring operation, you must call the Undo command from a Java source file. Though Undo affects deployment descriptors, the command cannot be called from a deployment descriptor.

For step-by-step descriptions of refactoring features, see Chapter 4. For notes on servlet refactoring, see Chapter 6. Following are some notes regarding the J2EE implications of some of the refactoring operations:

[lb] For CMP beans, the method rename of a selector or finder automatically updates the EJBQL (Enterprise JavaBeans Query Language) statements and query element in the deployment descriptor.. However, the IDE does not allow you to change the method name of a mandatory EJB method such as ejbCreate().

Even in the case where a renamed method was previously overloaded, a new deployment descriptor entry is made for the refactored method name. For instance, updates to deployment descriptors tagged as primary key fields and composite primary key fields as a result of method rename refactoring are supported. Also, a rename of web service endpoint interfaces are handled by the NetBeans IDE by updating the appropriate WSDL.

- [lb] Even in the case of a CMP bean, a field rename refactoring of a primary key results in field accessors deployment descriptors being updated. In addition, if a relationship field is renamed, the cmr-field in the deployment descriptor is updated.
- [lb] Encapsulating fields refactoring in an enterprise bean class works the same way as in standard classes, with the exception of primary keys in CMP enterprise beans, which cannot be refactored in NetBeans IDE 4.1.