Introducing EJB-CMP/CMR, Part 2 of 3

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. Introduction 2
2. Application design 6
3. Example 3: The one-to-one bidirectional relationship 8
4. Example 4: The many-to-many unidirectional relationship 21
5. Example 5: The one-to-many bidirectional relationship 27
6. Summary 35
7. Feedback 37

Section 1. Introduction

Introduction to CMP/CMR, Part 2

This is the second part of a tutorial series that is designed to introduce you to Container-Managed Persistence (CMP) and Container-Managed Relationships (CMR) in Enterprise JavaBeans 2.0 (EJB). These features are particular to EJB entity beans that are typically long-lived as compared to that are normally transitory.

Enterprise JavaBeans (EJB) 2.0 extends the earlier version 1.1 by adding advanced support for entity beans as follows:

- Updated container-managed persistence for entity beans
- Support for container-managed relationships
- EJB-Query Language (EJB-QL) for portable `select` and `find` query methods defined in the deployment descriptor
- The addition of local interfaces and local home interfaces to optimize access from other beans in the same container

If you want to buy or sell components, you will most likely want a layer of persistence in your components to work cross-platform on application servers (for example, IBM WebSphere, BEA WebLogic, JBoss/Tomcat, etc.) and persistence storage systems (for example, Oracle, DB2, etc.). You do not have to write low-level Java Database Connectivity (JDBC) calls in your EJBs to add these features, which is a great saver of time and complexity. Once you get the hang of CMP/CMR, it is faster to write entity beans using this technology, than using low-level JDBC inside of bean-managed persistence (BMP) beans .

This tutorial assumes that you have completed the first part of this three part series (see Resources on page 36). Again, if you are not familiar with EJB or you need to refresh your memory, I recommended that you read Enterprise JavaBeans Fundamentals, an IBM tutorial written by Richard Monson-Haefel and Tim Rohaly. The Enterprise JavaBeans Fundamentals is an excellent tutorial written by an excellent author.

Should I care about CMP/CMR?

What is the use of a container-managed entity bean? Well, for starters, you do not have to write low-level JDBC calls to save the state of the bean and you do not have to write code to manage relationships. It is all built into the EJB framework. Your interface to relationships is through the pervasive `java.util.Collection` and `java.util.Set` which most EJB developers are already familiar with. Very cool!

This additional feature includes support for JavaBeans component patterns for persistent fields, inside of the entity bean. Thus, instead of making your class variables public -- which has always felt strange to me -- you create get and set methods following the JavaBean technology standard naming pattern we all know and love.

I can't stress this point enough. Since EJB 2.0 containers will support the most common SQL databases (and other data stores as well), you can write components that work with many types of databases. This makes it easier to sell components that require persistent storage.

For example, you can sell components that will work in an IT department that uses Oracle or a shop that uses DB2. Thus, instead of writing low-level JDBC calls using SQL native to a particular database, you will use EJB-QL to create finder and select methods, and describe relationships in deployment descriptors.

Simply put, CMP/CMR is the missing link in cross-platform component creation. CMP/CMR will spur the growth of the enterprise level component marketplace. In addition, CMP/CMR is easier to use then low-level JDBC calls. CMP/CMR corrects many of the foibles, and missing functionality of earlier versions of CMP. There are many persistence frameworks, none are available on as many application server platforms as EJB CMP/CMR!

What do I need to know for this tutorial?

The example code in this tutorial is written to work with any J2EE compliant application server that supports EJB 2.0. The example code endeavors to be compliant; thus, all example code was deployed on the J2EE reference implementation that ships with Java 2 SDK, Enterprise Edition 1.3. The example code should deploy to your application server of choice by just modifying the Ant build scripts and corresponding deployment descriptors as long as your application server support EJB 2.0, and therefore, supports CMP/CMR.

This tutorial assumes that you are familiar with Java programming language and to some extent EJB; although, you do not have to be an expert. Since I will be covering deployment descriptors, which are written in XML, you should have a rudimentary knowledge of XML. If you are not familiar with EJB, I recommended that you read *Enterprise JavaBeans Fundamentals*, an developerWorks tutorial written by Richard Monson-Haefel and Tim Rohaly (See Resources on page 36). This is an excellent tutorial written by great authors. Even if you do not read this tutorial word for word, I suggest you at least use it as a reference.

Although not a prerequisite per se, knowledge of `Ant`, an XML-based, open source build system similar to make, will be helpful to understand the build scripts presented in the examples.

You do not need knowledge of JDBC since there will be no low-level calls in this tutorial, but basic knowledge of SQL and relational database theory is required.

What will this tutorial cover?

Instead of writing a giant tutorial that would take days to go through. I have split the tutorial into three parts that can each be finished in an hour or so. You could finish each tutorial during a lunch break, so get a sandwich and a beverage, and get started.

EJB 2.0 added a lot of features and functionality, this tutorial focuses on CMP/CMR. Thus, this tutorial assumes you have a background with EJB, and entity beans. You don't have to be an EJB expert to follow along. The tutorials cover local interfaces, deployment descriptor CMP, CMR fields, and relationship elements. I also cover the full range of relationship types as follows:

- One-to-one
- One-to-many

- Many-to-many

The relationships in the example also cover both unidirectional support and bidirectional support. The relationships are defined in XML deployment descriptors.

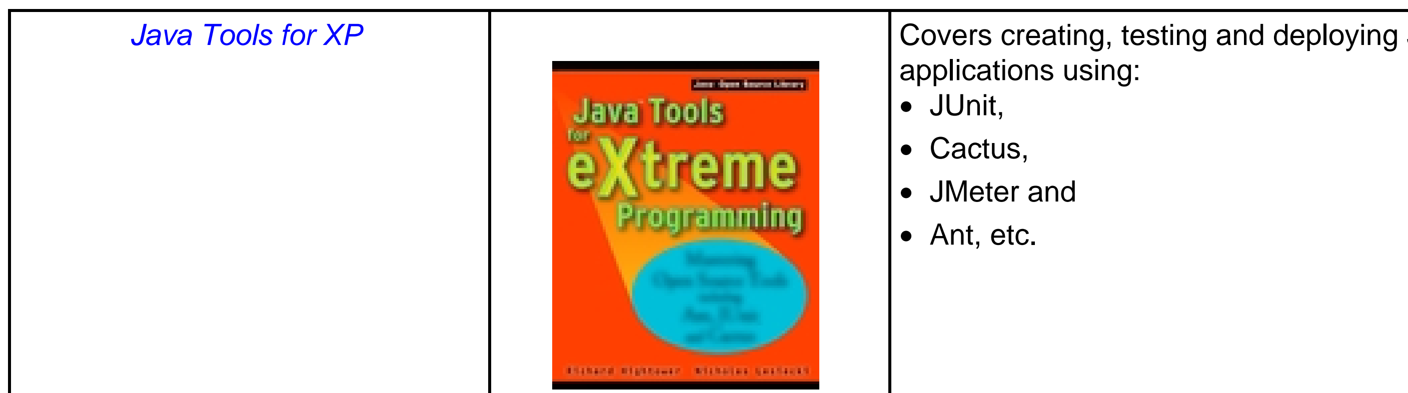
In the first tutorial, you get a taste of CMP/CMR and EJB-QL, with an example of a simple EJB 2.0 style CMP entity bean. Part of the example demonstrates simple EJB-QL to create a finder method without a Java implementation. This tutorial gets you acclimated to the terminology and technology, and adds a basic example.

In this tutorial, the second one in this series, you will build on the first example to cover each type of relationship and each type of relationship direction. You will create a a client that accessesthe relationships you created to add, remove and change related members.

About the Author

Rick Hightower, Director of Development at *eBlox*, has over a decade of experience as a software developer. He leads the adoption of new processes like Extreme Programming, and technology adoption like adoption of CMP and CMR.

Rick's publications include *Java Tools for eXtreme Programming*, which covers deploying and testing J2EE projects (published by *John Wiley)*, contributions to Java Distributed Objects (published by Sams), and several articles in *Java Developer's Journal.*



Java Tools for XP		Covers creating, testing and deploying applications using: • JUnit, • Cactus, • JMeter and • Ant, etc.

Also expect to see his book on Jython from Addison Wesley in the near future.

Rick has also taught classes on developing Enterprise JavaBeans, JDBC, CORBA, Applets as CORBA clients, etc.

Rick is also updating the next version of the Enterprise JavaBeans Developer's Guide to the 2.0 Specification for TriveraTech. The last version of this guide, which covered EJB 1.1 was

distributed to over 100,000 developers. This free guide discusses key EJB architectural concepts so developers can have a deeper understanding of EJB. The newest version of this guide will be released soon.

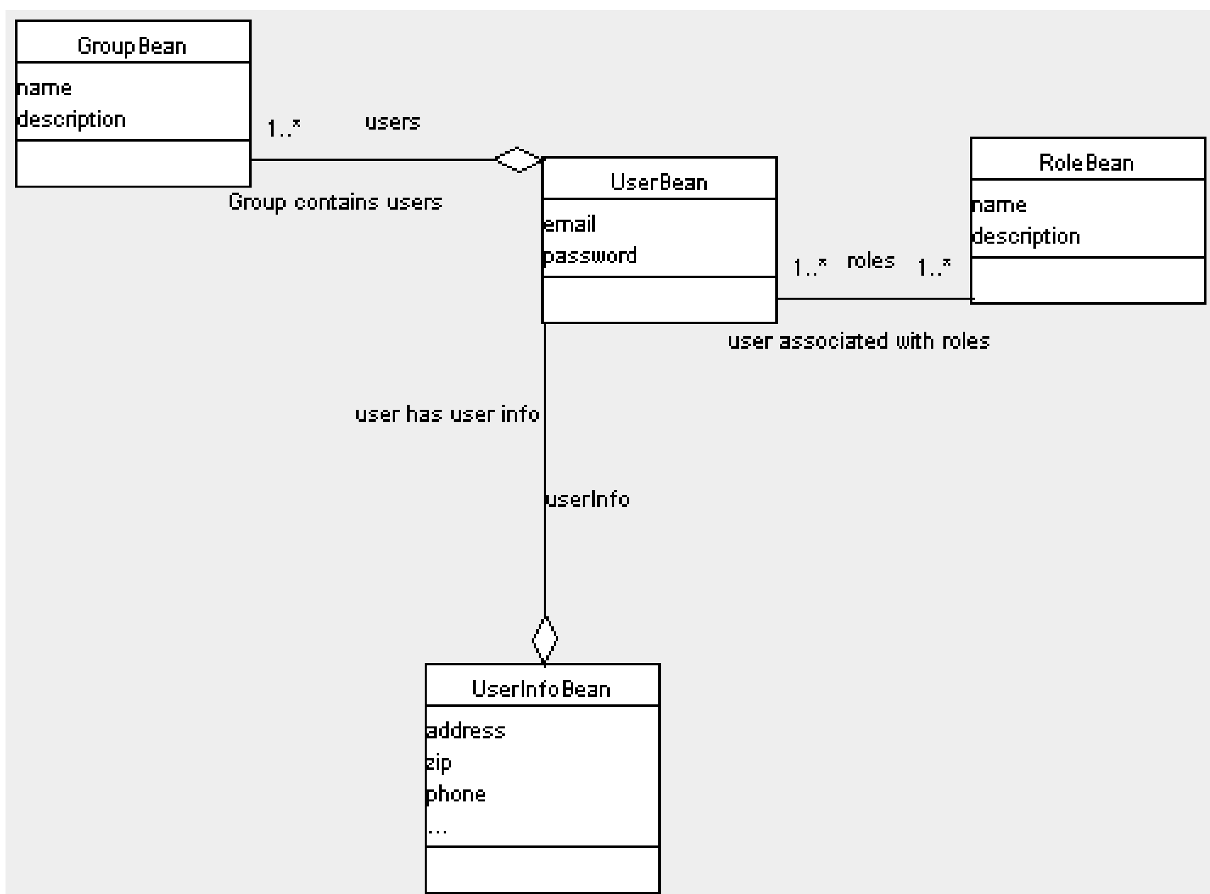
Section 2. Application design

Application design: Entity design

The examples in this tutorial series use a fictitious authentication subsystem for an online content management system that is designed to allow the following:

- log users into system
- authenticate users are in certain roles
- allow user to be organized into groups to allow group operations
- store user information like address and contact information
- manage roles, users, groups (manage = add, edit, delete)

Figure 1: Overview of the entities in this tutorial



An overview of the entities in the system in Figure 1 shows that there are four distinct entities: *User*, *Group*, *Role*, and *UserInfo*. Each of these entities have the following three relationships:

- Users are associated with Roles (many-to-many)
- A `User` has UserInfo (one-to-one)
- A Group contains Users (one-to-many)

The first tutorial showed how to build CMP into the UserManagement entity beans for this project. In this second tutorial, I will show how to implement each of these entites with their relationships.

Section 3. Example 3: The one-to-one bidirectional relationship

Relationships

Relationships have directions: unidirectional and bidirectional. And, relationships have `<multiplicity>`: one-to-one, one-to-many, and many-to-many.

The first example relationship is bidirectional with a `<multiplicity>` of one-to-one. The relationship will be between the `User` and his or her contact information stored in `UserInfo`. This example will do the following:

1. Define the `UserInfoBean` entity bean.
2. Add CMR fields to the `UserBean`.
3. Add a relationship element to the deployment descriptor.
4. Add code to the `UserManagement` session bean to use this relationship.
5. Add code to the client to use the new code in the `UserManagement` session bean.

Defining the `UserInfoBean`

The `UserInfoBean`, like the `UserBean`, defines CMP fields. The CMP fields are used to store contact information and other information describing the `UserBean`.

Thus the `UserInfoBean` defines a local interface, a local home interface, an entity bean implementation class, and a entity element entity in the deployment descriptor. All of these are shown in Listings 1 through 4.

Listing 1: UserInfoBean's Local interface

```
/* Local interface */
package com.rickhightower.auth;

import javax.ejb.EJBLocalObject;

public interface LocalUserInfo extends EJBLocalObject {
    public abstract String getEmail();

    public abstract LocalUser getUser();
    public abstract void setUser(LocalUser user);

    public abstract String getDept();
    public abstract void setDept(String value);

    public abstract String getWorkPhone();
    public abstract void setWorkPhone(String value);

    public abstract String getExtention();
    public abstract void setExtention(String value);

    public abstract boolean getEmployee();
    public abstract void setEmployee(boolean value);
```

```
    public abstract String getHomePhone();
    public abstract void setHomePhone(String value);

    public abstract String getFirstName();
    public abstract void setFirstName(String value);

    public abstract String getLastName();
    public abstract void setLastName(String value);

    public abstract String getMiddleName();
    public abstract void setMiddleName(String value);
}
```

Listing 2: UserInfoBean's Home interface

```
/* Local home interface */
package com.rickhightower.auth;

import javax.ejb.EJBLocalHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

import java.util.Collection;

public interface LocalUserInfoHome extends EJBLocalHome {

    public LocalUserInfo create(
                        String firstName, String middleName,
                        String lastName, String email,
                        String dept, String workPhone,
                        String extention, String homePhone,
                        boolean isEmployee)
                                    throws CreateException;

    public LocalUserInfo findByPrimaryKey (String email)
                                          throws FinderException;

}
```

Packaging `UserInfoBean`

Please note that entities involved in a relationship must be defined in the same deployment descriptor; thus, they must be packaged in the same EJB *.jar* file. The `UserBean` and the `UserInfoBean` are packaged together in the same *.jar* file and defined together in the same deployment descriptor.

Listing 3: UserInfoBean's implementation

```
/* entity bean implementation class */
package com.rickhightower.auth;

import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.CreateException;
import javax.naming.*;
```

```
public abstract class UserInfoBean implements EntityBean {

    public String ejbCreate(
                        String firstName, String middleName,
                        String lastName, String email,
                        String dept, String workPhone,
                        String extention, String homePhone,
                        boolean isEmployee)
                                    throws CreateException {
        setEmail(email);
        setDept(dept);
        setWorkPhone(workPhone);
        setExtention(extention);
        setHomePhone(homePhone);
        setEmployee(isEmployee);
        setFirstName(firstName);
        setLastName(lastName);
        setMiddleName(middleName);
        return null;
    }

    public void ejbPostCreate(
                        String firstName, String middleName,
                        String lastName, String email,
                        String dept, String workPhone,
                        String extention, String homePhone,
                        boolean isEmployee)
                                  throws CreateException { }

    public abstract String getEmail();
    public abstract void setEmail(String value);

    public abstract String getFirstName();
    public abstract void setFirstName(String value);

    public abstract String getLastName();
    public abstract void setLastName(String value);

    public abstract String getMiddleName();
    public abstract void setMiddleName(String value);

    public abstract LocalUser getUser();
    public abstract void setUser(LocalUser user);

    public abstract String getDept();
    public abstract void setDept(String value);

    public abstract String getExtention();
    public abstract void setExtention(String value);

    public abstract String getWorkPhone();
    public abstract void setWorkPhone(String value);

    public abstract boolean getEmployee();
    public abstract void setEmployee(boolean value);

    public abstract String getHomePhone();
    public abstract void setHomePhone(String value);

    public void setEntityContext(EntityContext context){ }
    public void unsetEntityContext(){ }
    public void ejbRemove(){ }
    public void ejbLoad(){ }
```

```
    public void ejbStore(){ }
    public void ejbPassivate(){ }
    public void ejbActivate(){ }
}
```

Also note that the following:

```
public abstract LocalUser getUser();
public abstract void setUser(LocalUser user);
```

is in both the local interface and the implementation of the `UserInfoBean`. This setter and getter method defines the CMR filed of this bidirectional relationship. Since the relationship is bidirectional, both `UserInfo` and `UserInfoBean` must have CMR fields referring to the other.

Listing 4: UserInfoBean's deployment descriptor

```
<entity>
   <display-name>UserInfoBean</display-name>
   <ejb-name>UserInfoBean</ejb-name>

   <local-home>com.rickhightower.auth.LocalUserInfoHome</local-home>
   <local>com.rickhightower.auth.LocalUserInfo</local>
   <ejb-class>com.rickhightower.auth.UserInfoBean</ejb-class>

   <persistence-type>Container</persistence-type>
   <prim-key-class>java.lang.String</prim-key-class>

   <reentrant>True</reentrant>
   <cmp-version>2.x</cmp-version>

   <abstract-schema-name>UserInfoBean</abstract-schema-name>

   <cmp-field>
     <field-name>firstName</field-name>
   </cmp-field>

   <cmp-field>
     <field-name>middleName</field-name>
   </cmp-field>

   <cmp-field>
     <field-name>lastName</field-name>
   </cmp-field>

   <cmp-field>
     <field-name>email</field-name>
   </cmp-field>

   <cmp-field>
     <field-name>dept</field-name>
   </cmp-field>

   <cmp-field>
     <field-name>workPhone</field-name>
   </cmp-field>

   <cmp-field>
     <field-name>extention</field-name>
```

```
    </cmp-field>

    <cmp-field>
      <field-name>homePhone</field-name>
    </cmp-field>

    <cmp-field>
      <field-name>employee</field-name>
    </cmp-field>

    <cmp-field>
      <field-name>email</field-name>
    </cmp-field>

    <primkey-field>email</primkey-field>


</entity>
```

Now you have to add the CMR fields to the `UserBean` in the next panel.

Adding CMR fields to `UserBean`

You need to add a CMR field to both the local interface and the entity bean class of the `UserBean`.

Defining a CMR field is much like defining a CMP field with the exception that the CMR field will return and pass the local interface of the other entity bean in the relationship from the getter and setter methods respectively. In this case the getter method will return `LocalUserInfo` as follows:

```
 public abstract LocalUserInfo getUserInfo();
```

and the setter method will pass the `LocalUserInfo` as follows:

```
 public abstract void setUserInfo(LocalUserInfo, userInfo);
```

Like the CMP fields the actual code to implement these methods are defined by the container implementation. All you have to do is define the relationship in the deployment descriptor and the EJB container will provide the code to manage the relationship. Sweet!

```
package com.rickhightower.auth;

import javax.ejb.EJBLocalObject;

public interface LocalUser extends EJBLocalObject {

    public String getEmail();
    public String getPassword();
    public LocalUserInfo getUserInfo();
    public void setUserInfo(LocalUserInfo userInfo);
}
```

```
public abstract class UserBean implements EntityBean {


    public abstract LocalUserInfo getUserInfo();
    public abstract void setUserInfo(LocalUserInfo userInfo);


}
```

Defining the relationships in the deployment descriptor

The relationships are defined outside of the `<enterprise-beans>` element. When you specify the relationship, you must specify both entity beans involved in the relationship. The relationship is defined in the `<ejb-relation>` element. Each role in the relationship is defined in the `<ejb-relationship-role>`.

Listing 5: Relationships for the entity beans

```
  <relationships>
    <ejb-relation>
      <ejb-relation-name>UserHasUserInfo</ejb-relation-name>

      <ejb-relationship-role>
        <ejb-relationship-role-name>
                  UserHasUserInfo
        </ejb-relationship-role-name>

        <multiplicity>One</multiplicity>
        <relationship-role-source>
          <ejb-name>UserBean</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>userInfo</cmr-field-name>
        </cmr-field>
      </ejb-relationship-role>

      <ejb-relationship-role>
        <ejb-relationship-role-name>
                  UserInfoPartOfUser
        </ejb-relationship-role-name>

        <multiplicity>One</multiplicity>
        <cascade-delete />
        <relationship-role-source>
          <ejb-name>UserInfoBean</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>user</cmr-field-name>
        </cmr-field>
      </ejb-relationship-role>
```

```
    </ejb-relation>
  </relationships>
```

The <ejb-relationship-role> has a name, multiplicity, source, and an optional CMR field defined by the following elements respectively: <ejb-relationship-role-name>, <multiplicity>, <relationship-role-source> with <ejb-name> sub-element, and the <cmr-field> with <cmr-field-name> sub-element, respectively.

Defining the relationships in the deployment descriptor

The <ejb-relationship-role-name> element body can be any name you wish. Try to make it descriptive for the relationship you are describing. Also, try to make it unique in the context of the deployment descriptor as follows:.

```
      <ejb-relationship-role>
        <ejb-relationship-role-name>UserHasUserInfo</ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
```

The <relationship-role-source>/<ejb-name> body must be set to the ejb-bean name as defined by the entity/<ejb-name> in the <enterprise-beans> element.

```
        <relationship-role-source>
          <ejb-name>UserBean</ejb-name>
        </relationship-role-source>
```

The <cmr-field>/<cmr-field-name> body must be set to the CMR field name as defined in the on page panel. If the <cmr-field> is defined by <getUserInfo()> and <setUserInfo()> in the local interface then it will be defined by <userInfo> in the <cmr-field-name> as follows:

```
        <cmr-field>
          <cmr-field-name>userInfo</cmr-field-name>
        </cmr-field>
      </ejb-relationship-role>
```

There are always two <ejb-relationship-role>s defined in an <ejb-relation>. If the relationship is bidirectional then both sides of the relationships will have a <cmr-field>. Here it the other side of the UserHasUserInfo relationship as follows:

```
      <ejb-relationship-role>
        <ejb-relationship-role-name>UserInfoPartOfUser</ejb-relationship-role-name>
```

```
        <multiplicity>One</multiplicity>
        <cascade-delete />
        <relationship-role-source>
          <ejb-name>UserInfoBean</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>user</cmr-field-name>
        </cmr-field>
      </ejb-relationship-role>
```

Here you can see that the name of this role is `UserInfoPartOfUser` (body of `<ejb-relationship-role-name>`). You can also see that the entity bean source is `UserInfoBean` (body of element `<relationship-role-source>`/`<ejb-name>`). And, you can pick out the CMR field involved in the relationship is user by looking at `<cmr-field>`/`<cmr-field-name>`.

Change `UserManagement` to use the relationship

To show how to work with CMR fields the `UserManagement` session bean adds three methods:

- a new version of `addUser()` that takes contact information
- a `getUsersInfo()` method that returns an array of value objects
- a `changeLastName()` method that uses the `<userInfo>` CMR field

Listing 6 and Listing 7 show the modified `UserManagement` bean.

Listing 6: UserManagement and UserManagementBean interface and implementation

```
/** Remote interface */
public interface UserManagement extends EJBObject {

    public void addUser(String email, String password,
                        String firstName, String middleName,
                        String lastName,  String dept,
                        String workPhone, String extention,
                        String homePhone, boolean isEmployee
    ) throws RemoteException;

    public UserValue[] getUsersInfo()throws RemoteException;

    public void changeLastName(String email, String lastName)
                                            throws RemoteException;




/** Entity Implementation class */
public class UserManagementBean implements SessionBean {

    public void addUser(String email, String password,
                        String firstName, String middleName,
```

```
                        String lastName,  String dept,
                        String workPhone, String extention,
                        String homePhone, boolean isEmployee){
        try {
            LocalUser user = userHome.create(email, password);
            LocalUserInfo info = infoHome.create(firstName,
                                  middleName, lastName, email, dept,
                                  workPhone, extention, homePhone,
                                  isEmployee);
            user.setUserInfo(info);
        } catch (CreateException e) {
            throw new EJBException
                  ("Unable to create the local user " + email, e);
        }

    }

    public UserValue[] getUsersInfo(){
        try{
            ArrayList userList = new ArrayList(50);
            Collection collection = userHome.findAll();
            Iterator iterator = collection.iterator();
            while(iterator.hasNext()){
               LocalUser user = (LocalUser)iterator.next();
               LocalUserInfo info = user.getUserInfo();
               UserValue userValue = new UserValue(user.getEmail());
               copyUserInfo(userValue, info);
               userList.add(userValue);
            }
            return (UserValue[])
                userList.toArray(new UserValue[userList.size()]);

        } catch (FinderException e){
            throw new EJBException
                        ("Unable to get list of users ", e);
        }
    }

    public void changeLastName(String email, String lastName){
        try {
            LocalUser user = userHome.findByPrimaryKey(email);
            LocalUserInfo info = user.getUserInfo();
            info.setLastName(lastName);
        } catch (FinderException e) {
            throw new EJBException
               ("Unable to change the last name " + lastName, e);
        }

    }
```

Listing 7: The Serializable UserValue class for UserManagementBean

```
/** Serializable value class */

package com.rickhightower.auth;

public class UserValue implements java.io.Serializable{

    private String email="";
    private String firstName="";
```

```
    private String middleName="";
    private String lastName="";
    private String dept="";
    private String workPhone="";
    private String extention="";
    private String homePhone="";
    private boolean employee;


    public UserValue() {
    }

    public UserValue(String aEmail) {
       email = aEmail;
    }


    public String getEmail() { return email;}
    public void setEmail(String email) {this.email=email;}


    public String getFirstName() {return firstName;}
    public void setFirstName(String firstName){
       this.firstName=firstName;
    }

    public String getMiddleName() {return middleName;}
    public void setMiddleName(String middleName){
       this.middleName=middleName;
    }

    public String getLastName() {return lastName;}
    public void setLastName(String lastName) {
       this.lastName=lastName;
    }

    public String getDept() {return dept;}
    public void setDept(String dept) {
       this.dept=dept;
    }

    public String getWorkPhone() {return workPhone;}
    public void setWorkPhone(String workPhone) {
       this.workPhone=workPhone;
    }

    public String getExtention() {return extention;}
    public void setExtention(String extention) {
       this.extention=extention;
    }

    public String getHomePhone() {return homePhone;}
    public void setHomePhone(String homePhone) {
       this.homePhone=homePhone;
    }

    public boolean isEmployee() {return employee;}
    public void setEmployee(boolean employee) {
       this.employee = employee;
    }

}
```

The new UserManagement methods

addUser()

The addUser() method demonstrates setting a CMR field.

The addUser() method takes all of the contact information and the user information as arguments. It then creates a LocalUser and a LocalUserInfoinstance using the home interfaces of both the UserBean and the UserInfoBean as follows:

```
public void addUser(String email, String password, boolean isEmployee){

    LocalUser user = userHome.create(email, password);
      LocalUserInfo info = infoHome.create(firstName, middleName, lastName,
        email, dept, workPhone, extention, homePhone, isEmployee);
```

Once both the LocalUser and LocalUserInfo object are created, the addUser() method sets the <userInfo> CMR field of the User with the newly create UserInfo as follows:

```
user.setUserInfo(info);
```

getUsersInfo()

The <getUserInfo()> method demonstrates getting a CMR field.

Since you can not return a collection of local entity from a remote method, the <getUserInfo()> demonstrates how to return a number entity beans with value objects.

Similar to the getUsers method, the getUsersInfo() method uses the findAll method of the

UserBean's home interface (LocalUserHome). However, instead of returning the ID of the UserBean (the e-mail) for each user, it returns a value object for each user that contains the department, e-mail address, employment status, phone with extention, last name, middle name, and first name of each user as follows:

```
ArrayList userList = new ArrayList(50);
   Collection collection = userHome.findAll();
   Iterator iterator = collection.iterator();
   while(iterator.hasNext()){
     LocalUser user = (LocalUser)iterator.next();
     LocalUserInfo info = user.getUserInfo();
        UserValue userValue = new UserValue(user.getEmail());
```

```
        copyUserInfo(userValue, info);
            userList.add(userValue);
             }
           return (UserValue[]) userList.toArray(new UserValue[userList.size()]);
```

The `UserValue` class is populate for each `User/UserInfo` pair.

The `UserValue` class is a simple serliazable class listed left. The `getUsersInfo()` method call a helper method called `copyUserInfo()` for each `User/UserInfo` pair. The `copyUserInfo()` copies the values from the `User/UserInfo` pair into a `UserValue` object. The `copyUserInfo()` method is listed below.

```
private void copyUserInfo(UserValue userValue, LocalUserInfo info){
    if (info != null){
   userValue.setFirstName(info.getFirstName());
       userValue.setMiddleName(info.getMiddleName());
       userValue.setLastName(info.getLastName());
          userValue.setDept(info.getDept());
          userValue.setWorkPhone(info.getWorkPhone());
          userValue.setExtention(info.getExtention());
          userValue.setHomePhone(info.getHomePhone());
          userValue.setEmployee(info.getEmployee());
                     }
                          }
```

changeLastName()

The `changeLastName()` method demonstrates accessing the CMR field of an entity bean to modify one of the related objects CMP fields.

The `changeLastName()` method uses the e-mail address passed to it to look up the `UserBean` local interface `LocalUser` and then use the `LocalUser`'s CMR field to gain access to the `UserInfoBean`. It then modifies the last name of the `UserInfoBean` all shown as follows:

```
LocalUser user = userHome.findByPrimaryKey(email);
LocalUserInfo info = user.getUserInfo();
     info.setLastName(lastName);
```

The next section will go over the changes to the client to exercise these three new methods.

Change the client to use new `UserManagement` methods

Just as before you have to change the client to use the new methods defined in the `UserManagement` bean.

Note the code below uses the new `addUser()`, `changeLastName()`, and `getUsersInfo()` methods.

In fact you create two users with addUsers. Then you change the last name of the user *Andy*. And verify the name change went through by printing out the *lastName* by utilizing the `getUsersInfo()` method of the `UserManagementBean`. Serendipity!

```
/* Add some user with the new addUser method */
 userMgmt.addUser("andy@rickhightower.com", "starwars",
              "Andy", "Mike", "Barfight",
               "Engineering", "555-1212", "x102",
              "555-5555", true);
 userMgmt.addUser("donna@rickhightower.com", "cusslikeasailor",
              "Donna", "Marie", "Smith",
              "Marketing", "555-1213", "x103",
              "555-7777", true);


 System.out.println("Change last name");
 userMgmt.changeLastName("andy@rickhightower.com", "Smith-Barfight");


 System.out.println("Get User Info");
 UserValue [] users = userMgmt.getUsersInfo();

 for (int index=0; index < users.length; index++){
  UserValue user = users[index];
  System.out.println("user firstName =" + user.getFirstName());
  System.out.println("user lastName =" + user.getLastName());
  System.out.println("user homePhone =" + user.getHomePhone());
 }
```

Compiling and deploying the one-to-one example

The same technique can be used to the build, package, deploy, and run the client. Please refer to the section 8 in the first installment of this tuturial (see Resources on page 36) to refresh your memory how to build and run this example application. Look for the source and build file under *cmpCmr/section3*.

Be sure to run `ant clean` to delete the old examples intermediate build files.

On the home stretch now!

You are done with the first three examples. If you made it this far, you are speeding to the finish line. The next section will cover many to many relationships.

Section 4. Example 4: The many-to-many unidirectional relationship

Many-to-many relationship

A many to many relationship sound pretty tough to implement. Actually, it is just as easy to implement as a one-to-one relationship. The EJB container does all of the hard work for you.

For this example, you will add different roles that a user can be in. An individual user can be in more than one role. And a role can be associated with more than one user.

In this example you will do the following:

1. Define the `RoleBean`.
2. Define the relationship from `UserBean` to `RoleBean` in the deployment descriptor.
3. Add a CMR collection field to the `UserBean`.
4. Add code to the `UserManagement` session bean to use this relationship.
5. Add code to the client to use the new code in the `UserManagement` session bean.

Define the `RoleBean`

The `RoleBean` is similar to the `UserInfoBean` in the last example. Like all of the other entity bean., it has CMP fields. Also it has a local home, a local interface, and entity bean implementation.

Unlike the `UserInfoBean` in the last example, the `RoleBean` does not have a `<cmr-field>`. The complete listings for the `RoleBean` is listed below.

Listing 8: `RoleBean` Local interface

```
package com.rickhightower.auth;
import javax.ejb.EJBLocalObject;

public interface LocalRole extends EJBLocalObject {
    public abstract String getName();
    public abstract String getDescription();
}
```

Listing 9: `RoleBean` Home interface

```
package com.rickhightower.auth;
import javax.ejb.EJBLocalHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.util.Collection;

public interface LocalRoleHome extends EJBLocalHome {
    public LocalRole create (String name, String description) throws CreateException;
    public LocalRole findByPrimaryKey (String email) throws FinderException;
```

```
    public Collection findAll() throws FinderException;
}
```

Listing 10: `RoleBean` EntityBean implementation class

```
package com.rickhightower.auth;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.CreateException;
import javax.naming.*;

public abstract class RoleBean implements EntityBean {

    public String ejbCreate(String name, String description) throws CreateException {
        setName(name);
        setDescription(description);
        return null;
    } public

    void ejbPostCreate(String name, String description) throws CreateException{ }

    public abstract String getName();
    public abstract void setName(String name);

    public abstract String getDescription();
    public abstract void setDescription(String description);

    public void setEntityContext(EntityContext context){ }
    public void unsetEntityContext(){ }
    public void ejbRemove(){ }
    public void ejbLoad(){ }
    public void ejbStore(){ }
    public void ejbPassivate(){ }
    public void ejbActivate(){ }
}
```

Listing 11: `RoleBean` deployment descriptor entry

```
    <entity>
      <display-name>RoleBean</display-name>
      <ejb-name>RoleBean</ejb-name>

      <local-home>com.rickhightower.auth.LocalRoleHome</local-home>
      <local>com.rickhightower.auth.LocalRole</local>
      <ejb-class>com.rickhightower.auth.RoleBean</ejb-class>

      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>

      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>

      <abstract-schema-name>RoleBean</abstract-schema-name>

      <cmp-field>
        <description>no description</description>
        <field-name>name</field-name>
      </cmp-field>
      <cmp-field>
```

```
        <description>no description</description>
        <field-name>description</field-name>
      </cmp-field>
      <primkey-field>name</primkey-field>
      <security-identity>
        <description></description>
        <use-caller-identity></use-caller-identity>
      </security-identity>

      <query>
        <description></description>
        <query-method>
          <method-name>findAll</method-name>
          <method-params />
        </query-method>
        <ejb-ql>select Object(o) from RoleBean o</ejb-ql>
      </query>

    </entity>

  </enterprise-beans>
```

Define the relationship in the deployment descriptor

The XML elements and technique for adding a many-to-many relationship is nearly identical as doing the one-to-one relationship. The only key difference is the multiplicity. The `<ejb-relationship-role>` for the `RoleBean` does not have a CMR field element because the relationship is unidirectional from the `UserBean` to the `RoleBean`. The `RoleBean` does not have knowledge of the `UserBean`. The `<ejb-relation>` element is listed as follows:

```
    <ejb-relation>
      <ejb-relation-name>UserAssociateWithRoles</ejb-relation-name>

      <ejb-relationship-role>
        <ejb-relationship-role-name>UserBeanToRoleBean</ejb-relationship-role-name>
        <multiplicity>Many</multiplicity>
        <relationship-role-source>
          <ejb-name>UserBean</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>roles</cmr-field-name>
          <cmr-field-type>java.util.Collection</cmr-field-type>
        </cmr-field>
      </ejb-relationship-role>

      <ejb-relationship-role>
        <ejb-relationship-role-name>RoleBeanToUserBean</ejb-relationship-role-name>
        <multiplicity>Many</multiplicity>
        <relationship-role-source>
          <ejb-name>RoleBean</ejb-name>
        </relationship-role-source>
      </ejb-relationship-role>

    </ejb-relation>
```

Notice that the `UserBeanToRoleBean` has a `<multiplicity>` of Many, and the `<cmr-field>` of `UserBeanToRoleBean` has a sub-element `<cmr-field-type>`. The `<cmr-field-type>` element defines the Java code type of the relationship with a `<multiplicity>` of Many. Note that the two possibilities are `java.util.Collection` and `java.util.Set`. Since you picked `java.util.Collection` and the CMR field name is roles, then you can expect to see a CMR field called roles (setter and getter method pair) in the local interface of the `UserBean` that returns and sets a `java.util.Collection`.

Add collection CMR field to `UserBean`

The CMR fields return many roles and set many roles. Thus unlike the CMR fields for the one-to-one relationship for `UserInfo`, you are going to set and get a collection of roles as follows:

Listing 12: `UserBean` EntityBean Implementation Class

```
public abstract class UserBean implements EntityBean {


    public abstract Collection getRoles();
    public abstract void setRoles(Collection roles);

}
```

Listing 13: `UserBean` Local interface

```
package com.rickhightower.auth;
import javax.ejb.EJBLocalObject;
import java.util.Collection;

public interface LocalUser extends EJBLocalObject {

    public Collection getRoles();
    public void setRoles(Collection roles);

    public String getEmail();
    public String getPassword();
    public LocalUserInfo getUserInfo();
    public void setUserInfo(LocalUserInfo userInfo);
}
```

That's it. It is not much different then the one-to-one relationship. Next I will show the methods in the `UserManagement` session bean that use the many-to-many relationship.

Use many-to-many relationship in the `UserManagement` bean

The `UserManagementBean` adds four methods to operate on Roles and the relationship between `User` and roles as follows: `inRole()`, `addRole()`, `createRole()`, and

`removeRole()`.

The `createRole()` and `removeRole()` methods are just boiler plate code to add and remove roles into the datastore -- nothing new.

addRole()

The `addRole()` adds a role to a `UserBean` as follows:

```
public void addRole(String email, String roleName){

    LocalUser user = userHome.findByPrimaryKey(email);
    LocalRole role = roleHome.findByPrimaryKey(roleName);
    Collection roles = user.getRoles();
    roles.add(role);

}
```

Notice that the `addRole()` looks up the role with the *roleHome*, then it adds the role to the collection from the CMR roles field. Thus, when a role gets added to the roles collection, the EJB container updates the datastore. Thus, the code that inserts the relationship in the datastore is the interface of the Collection class. Cool!

inRole()

The `inRole()` method checks to see if a user is in a certain role:

```
public boolean inRole(String email, String roleName){

    LocalUser user = userHome.findByPrimaryKey(email);
    Collection roles = user.getRoles();
    Iterator iter = roles.iterator();
    while (iter.hasNext()){
        LocalRole role = (LocalRole)iter.next();
        if (role.getName().equals(roleName)) return true;
    }

    return false;
}
```

Because the API for the datastore is the same as `java.util.Collection`, coding with CMR is easy and fast. In the next panel I will cover accessing these method from the client.

Using `UserManagement` `inRole()` and `addRole()` methods

The client uses all of the new `UserManagement` methods as follows:

```
    //Create Roles -- section 4
userMgmt.createRole("admin", "Adminstrator");
userMgmt.createRole("manager", "Content Manager");
userMgmt.createRole("user", "Normal User");
userMgmt.createRole("guest", "Guest User");

    //Add roles to user Andy -- section 4
userMgmt.addRole("andy@rickhightower.com", "admin");
userMgmt.addRole("andy@rickhightower.com", "user");

    //See if Andy is in role admin -- section 4
if (userMgmt.inRole("andy@rickhightower.com","admin")){
    System.out.println("Andy is an admin");
}

    //See if Andy is in role manager -- section 4
if (!userMgmt.inRole("andy@rickhightower.com","manager")){
    System.out.println("Andy is not a manager");
}

    /* Remove Roles -- section 4 */
userMgmt.removeRole("admin");
userMgmt.removeRole("manager");
userMgmt.removeRole("user");
userMgmt.removeRole("guest");
```

Compiling and deploying the many-to-many example

The same technique can be used to the build, package, deploy, and run the client. Please refer to section 8 in the first installment of this tuturial (see Resources on page 36) to refresh your memory how to build and run this example application. Look for the source and build file under *cmpCmr/section4*.

Be sure to run `ant clean` to delete the old examples intermediate build files.

Final lap

You are done with the first four examples. If you made it this far, it's all down hill from here and the wind is at your back. The next section will cover one-to-many relationships.

Section 5. Example 5: The one-to-many bidirectional relationship

One-to-many relationship

A one-to-many relationship is a lot like the other relationships. Just as before the EJB container does most of the heavy lifting.

In this example you will do the following:

1. Define the `GroupBean`.
2. Define the bidirectional relationship from `UserBean` to `GroupBean` in the deployment descriptor.
3. Add a CMR collection field to the `UserBean`.
4. Add code to the `UserManagement` session bean to use this relationship.
5. Add code to the client to use the new code in the `UserManagement` session bean.

Define the `GroupBean`

The `GroupBean` is similar to the `UserInfoBean` and the `RoleBean` in the last two examples. Like all of the other entity beans, it has CMP fields. Also it has a local home, a local interface, and entity bean implementation.

Like the `UserInfoBean` in the first relationship example, the `GroupBean` has a `<cmr-field>` referring to the `UserBean`. Unlike the `UserInfoBean`, the `<cmr-field>` users, in the `GroupBean` is a collection of `UserBean`s. The complete listing for the `GroupBean` is listed below.

Listing 14: `GroupBean` Local interface

```
package com.rickhightower.auth;
import javax.ejb.EJBLocalObject;
import java.util.Collection;

public interface LocalGroup extends EJBLocalObject {

    public abstract Collection getUsers();
    public abstract void setUsers(Collection collection);

    public abstract String getName();
    public abstract String getDescription();
}
```

Listing 15: `GroupBean` Home interface

```
package com.rickhightower.auth;

import javax.ejb.EJBLocalHome;
import javax.ejb.CreateException;
```

```
import javax.ejb.FinderException;
import java.util.Collection;

public interface LocalGroupHome extends EJBLocalHome {
    public LocalGroup create (String name, String description) throws CreateException;
    public LocalGroup findByPrimaryKey (String name) throws FinderException;

    public Collection findAll() throws FinderException;
}
```

Listing 16: `GroupBean` EntityBean implementation Class

```
package com.rickhightower.auth;

import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.CreateException;
import java.util.Collection;
import javax.naming.*;

public abstract class GroupBean implements EntityBean {

    public String ejbCreate(String name, String description) throws CreateException {
        setName(name);
        setDescription(description);
        return null;
    }

    public void ejbPostCreate(String name, String description) throws CreateException{ }

    public abstract Collection getUsers();
    public abstract void setUsers(Collection collection);

    public abstract String getName();
    public abstract void setName(String name);

    public abstract String getDescription();
    public abstract void setDescription(String description);

    public void setEntityContext(EntityContext context){ }
    public void unsetEntityContext(){ }
    public void ejbRemove(){ }
    public void ejbLoad(){ }
    public void ejbStore(){ }
    public void ejbPassivate(){ }
    public void ejbActivate(){ }
}
```

Listing 17: `GroupBean` deployment descriptor entry

```
    <entity>
      <display-name>GroupBean</display-name>
      <ejb-name>GroupBean</ejb-name>

      <local-home>com.rickhightower.auth.LocalGroupHome</local-home>
      <local>com.rickhightower.auth.LocalGroup</local>
      <ejb-class>com.rickhightower.auth.GroupBean</ejb-class>

      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
```

```
    <reentrant>False</reentrant>
    <cmp-version>2.x</cmp-version>

    <abstract-schema-name>GroupBean</abstract-schema-name>

    <cmp-field>
      <description>The group name</description>
      <field-name>name</field-name>
    </cmp-field>
    <cmp-field>
      <description>The description of the group</description>
      <field-name>description</field-name>
    </cmp-field>

    <primkey-field>name</primkey-field>

    <security-identity>
      <description></description>
      <use-caller-identity></use-caller-identity>
    </security-identity>

    <query>
      <description></description>
      <query-method>
        <method-name>findAll</method-name>
        <method-params />
      </query-method>
      <ejb-ql>select Object(group) from GroupBean group</ejb-ql>
    </query>

  </entity>
```

Define the relationship in the deployment descriptor

The xml elements and technique for adding a one-to-many relationship is nearly identical as doing the one-to-one and the many-to-many relationship. The only key difference is the multiplicity. The `<ejb-relationship-role>` for the `GroupBean` has a CMR field element because the relationship is bidirectional from the `UserBean` to the `GroupBean`. The `GroupBean` has a CMR field name users. The `UserBean` has a CMR field named group. The `<ejb-relation>` element is listed as follows:

```
  <ejb-relation>

    <ejb-relation-name>GroupsHaveUsers</ejb-relation-name>

    <ejb-relationship-role>
      <ejb-relationship-role-name>GroupHasUsers</ejb-relationship-role-name>
      <multiplicity>One</multiplicity>

      <relationship-role-source>
        <ejb-name>GroupBean</ejb-name>
      </relationship-role-source>

      <cmr-field>
        <cmr-field-name>users</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
```

```
      </ejb-relationship-role>

      <ejb-relationship-role>

        <ejb-relationship-role-name>UsersInGroup</ejb-relationship-role-name>
        <multiplicity>Many</multiplicity>

        <relationship-role-source>
          <ejb-name>UserBean</ejb-name>
        </relationship-role-source>

        <cmr-field>
          <cmr-field-name>group</cmr-field-name>
        </cmr-field>


      </ejb-relationship-role>

    </ejb-relation>
```

Notice that the `GroupHasUsers` has a multiplicity of *One* since there is one group in the relationship, and like the last example the `<cmr-field>` of `GroupHasUsers` has a sub-element `<cmr-field-type>` as follows:

```
        <cmr-field>
          <cmr-field-name>users</cmr-field-name>
          <cmr-field-type>java.util.Collection</cmr-field-type>
        </cmr-field>
```

The type of the field relationship is `java.util.Collection` and the CMR field name is users.

The other side of the relationship UsersInGroup has a `<multiplicity>` of Many since there are many users in the group. Also notice that the `<cmr-field>` is set to group.

Add CMR field group to `UserBean`

The CMR fields group allows you to set the user into a group or get the group that the user is in. Thus like the CMR fields for the one-to-one relationship for `UserInfo`, you can set and get a collection the associated group:

Listing 18: `UserBean` EntityBean implementation Class

```
public abstract class UserBean implements EntityBean {


    public abstract LocalGroup getGroup();
    public abstract void setGroup(LocalGroup group);

}
```

Listing 19: `UserBean` Local interface

```
package com.rickhightower.auth;

import javax.ejb.EJBLocalObject;
import java.util.Collection;

public interface LocalUser extends EJBLocalObject {

    public LocalGroup getGroup();
    public void setGroup(LocalGroup group);

    public Collection getRoles();
    public void setRoles(Collection roles);

    public String getEmail();
    public String getPassword();
    public LocalUserInfo getUserInfo();
    public void setUserInfo(LocalUserInfo userInfo);
}
```

It is not much different than the other relationship examples. Next I will show the methods in the `UserManagement` session bean that use the many-to-many relationship.

Use one-to-many relationship in the `UserManagement` bean

The `UserManagementBean` adds four methods each to operate on `Groups`, and the relationship between `User` and `Groups` as follows: `inGroup()`, `moveUserToGroup()`, `addRoleToUsers()`, `getUsersInGroup()`, `createGroup()`, `removeGroup()`, `getGroups()`, and `getRoles()`.

The `createGroup()` and `removeGroup()` methods are just boiler plate code to add and remove groups into the datastore -- again, nothing new. The methods `getRoles()` and `getGroups()` were added to show the available roles and group -- more boiler plate code.

`inGroup()`

The `inGroup()` method checks to see if a user is in a certain group:

```
public boolean inGroup(String email, String groupName){

    LocalUser user = userHome.findByPrimaryKey(email);
    return user.getGroup().getName().equals(groupName);

}
```

`moveUserToGroup()`

The `moveUserToGroup()` moves the user into a different group:

```
public void moveUserToGroup(String email, String groupName){
```

```
    LocalUser user = userHome.findByPrimaryKey(email);
    LocalGroup group = groupHome.findByPrimaryKey(groupName);

    user.setGroup(group);

        /* The relationship works both way so this would work too! */

    //<emph>group.getUsers().add(user);
```

addRoleToUsers()

The `addRoleToUsers()` utilizes both the group and the role relationships to the user. It adds a Role to each user in a group.

```
public void addRoleToUsers(String groupName, String roleName){

    LocalGroup group = groupHome.findByPrimaryKey(groupName);
    LocalRole role = roleHome.findByPrimaryKey(roleName);
    Collection users = group.getUsers();
    Iterator iterator = users.iterator();

    while(iterator.hasNext()){
        LocalUser user = (LocalUser)iterator.next();
        user.getRoles().add(role);
    }

}
```

getUsersInGroup()

The `getUsersInGroup()` uses both the user to `<userInfo>` relationship and the user to group relationship to return an array of strings that each contain the first name, last name, and e-mail address of each user in the group.

```
public String[] getUsersInGroup(String groupName){
    LocalGroup group = groupHome.findByPrimaryKey(groupName);
    Collection users = group.getUsers();
    Iterator iterator = users.iterator();
    ArrayList userList = new ArrayList (50);
    while(iterator.hasNext()){
        LocalUser user = (LocalUser)iterator.next();

        String firstName = user.getUserInfo().getFirstName();
        String lastName = user.getUserInfo().getLastName();
        String email = user.getEmail();

        StringBuffer sUser = new StringBuffer(80);
        sUser.append(firstName + ", ");
        sUser.append(lastName + ", ");
        sUser.append(email);
        userList.add(sUser.toString());
    }
    return (String []) userList.toArray(new String[userList.size()]);

}
```

Now let's modify the client to use this code.

Using `UserManagement` `inGroup()`, `moveUserToGroup()`, `addRoleToUsers()`, and `getUsersInGroup()` methods

The client uses all of the new `UserManagement` methods as follows:

```
System.out.println("Create and Display groups");
userMgmt.createGroup("marketing", "Marketing group");
userMgmt.createGroup("engineering", "Engineering group");
userMgmt.createGroup("sales", "Sales group");
userMgmt.createGroup("IT","Information Technology group");

    /** Display the created groups */
String [] groups = userMgmt.getGroups();

for (int index=0; index < groups.length; index++){
    System.out.println(groups[index]);
}

createUsers(userMgmt); //updated for section 3

    /** Section 5 Add role to users */
String group = "engineering";
String role = "super_user";
userMgmt.addRoleToUsers(group,role);

group = "IT";
role = "admin";
userMgmt.addRoleToUsers(group,role);

group = "marketing";
role = "user";
userMgmt.addRoleToUsers(group,role);


    /** Section 5 -- list users in group engineering */
String [] users = userMgmt.getUsersInGroup("engineering");
System.out.println("Users in the engineering group");

for (int index=0; index < users.length; index++){
     System.out.println(users[index]);
}

    /** Section 5 -- Is Andy in group Engineering ? */

if (userMgmt.inGroup("andy@rickhightower.com","engineering")){
    System.out.println("Andy is in the engineering group");
}

    /** Section 5 --
      Is Andy in group the super_user role that was assigned
      to all users in engineering? */
if (userMgmt.inRole("andy@rickhightower.com","super_user")){
    System.out.println("Andy is a super_user");
}

/** Section 5 -- Move Andy to marketing */
userMgmt.moveUserToGroup("andy@rickhightower.com", "marketing");
```

```
/** Section 5 -- Is Andy in group Engineering ? */
if (userMgmt.inGroup("andy@rickhightower.com","engineering")){
    System.out.println("Andy is in the engineering group");
} else if(userMgmt.inGroup("andy@rickhightower.com","marketing")){
    System.out.println("Andy is now in the marketing group");
}

/**Section 5 **/
userMgmt.removeGroup("marketing");
userMgmt.removeGroup("engineering");
userMgmt.removeGroup("sales");
userMgmt.removeGroup("IT");
```

Compiling and deploying the one-to-many example

The same technique can be used to the build, package, deploy and run the client. Please refer to section 8 in the first installment of this tuturial (see Resources on page 36) to refresh your memory how to build and run this example application. Look for the source and build file under *cmpCmr/section5*.

Be sure to run `ant clean` to delete the old example's intermediate build files.

Section 6. Summary

Done

You are now are done with the first five examples and the first two tutorials of this series. Looks like you made it.

You've created one-to-one, one-to-many, and many-to-many relationships. You have used the collection to add entities to other entities while the EJB Container adds the rows to the underlying join tables and fields.

For summary, you can view the relationships in the entity *.jar* file with the `deploytool` pictured in Figure 2 and Figure 3.

Figure 2: All the relationships you created in this tutorial

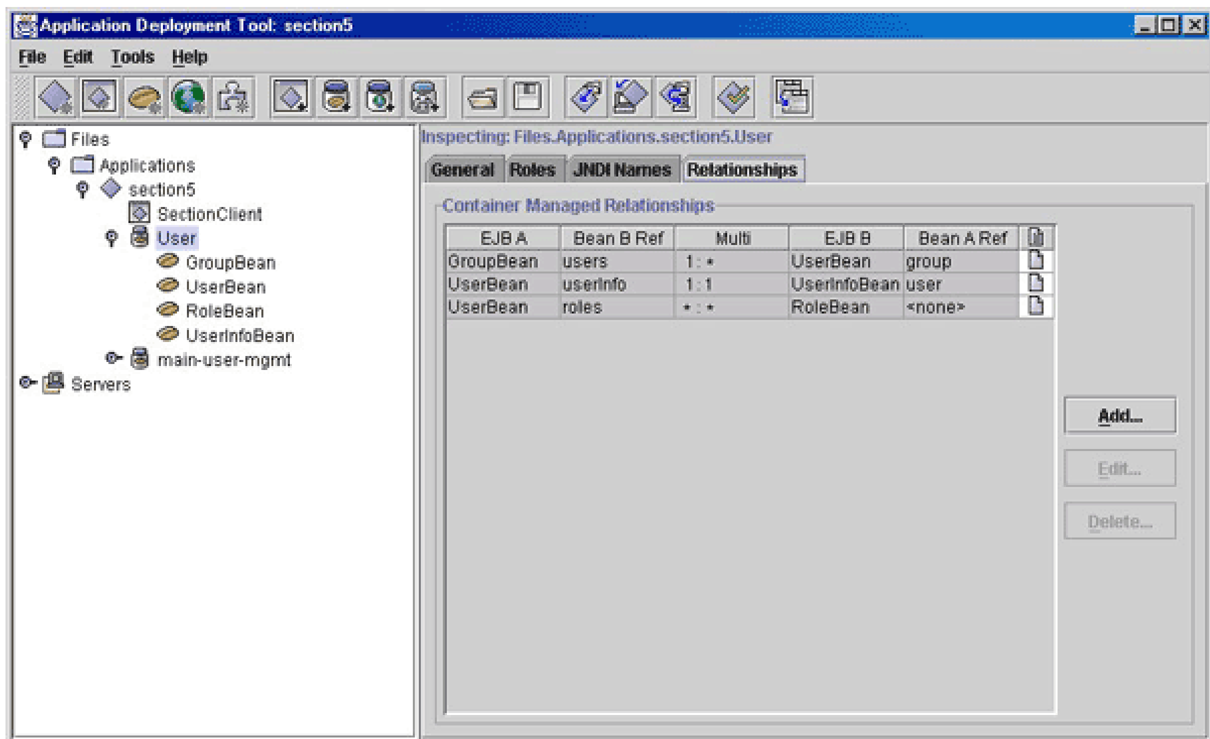
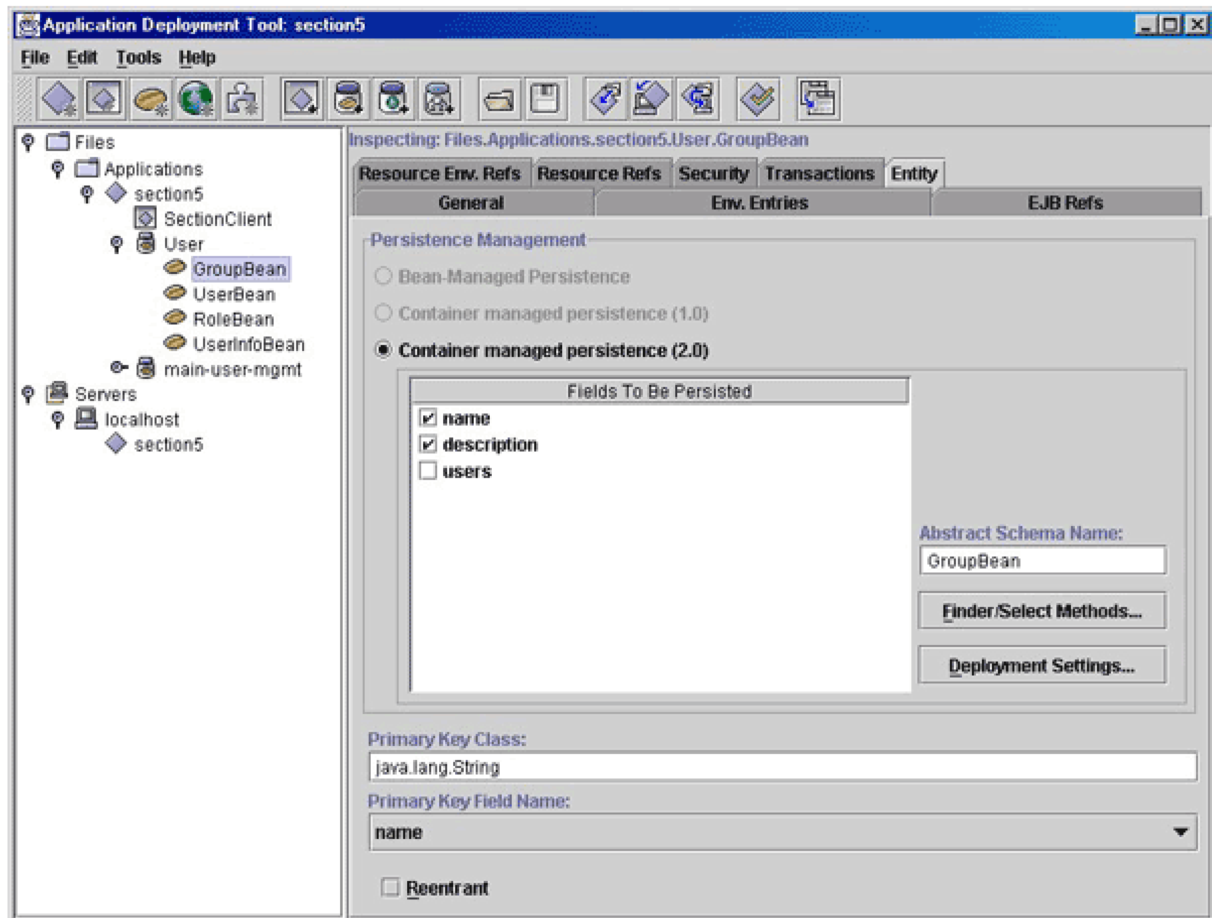


Figure 3: The CMR and CMP fields of Group



In the third and last tutorial of this series I will explain the power of EJB-QL.

Resources

- Here is a *zip file* with all the code.
- The first tutorial in this series *Introduction to CMP/CMR, Part 1*
- A good *tutorial on EJB CMP/CMR and EJB-QL*
- The *J2EE tutorial* from Sun
- *Developer's Guide to Understanding EJB 2.0 (updated by Rick Hightower)*
- *Enterprise JavaBeans fundamentals*

Books:

- *Mastering Enterprise JavaBeans (2nd Edition)* by Ed Roman, Scott W. Ambler, Tyler Jewell, Floyd Marinescu

  The EJB Encyclodpedia!
- *Enterprise JavaBeans (3rd Edition)* by Richard Monson-Haefel

  Get this one too!
- *Java Tools for Extreme Programming* by Richard Hightower, Nicholas Lesiecki

  Covers building and deploying J2EE applications with EJBs.

Section 7. Feedback

Feedback

Please let us know whether this tutorial was helpful to you and how I could make it better. I'd also like to hear about other tutorial topics you'd like to see covered. Thanks!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.