Edition

2

Date: 2002-06-10, 11:06:28 AM

DAIN SUNDSTROM

The JBoss Group

JBossCMP

DAIN SUNDSTROM AND THE JBOSS GROUP

JBossCMP

© JBoss Group, LLC
2520 Sharondale Dr.
Atlanta, GA 30305 USA
sales@jbossgroup.com

Table of Contents

Table of Listings

Table of Tables

Preface

Forward

In high school, I told my English teacher that mathematics and physics were much more important than his class. He smiled and laughed at my lame excuse for not doing an assignment and gave me a zero. Over the last year, I have learned how wrong I was. It doesn't matter how good your code is if no one can figure out how to use it.

About the Authors

Dain Sundstrom is the Chief Architect of JBossCMP, an implementation of the Enterprise Java Beans 2.0 Container Managed Persistence specification. Dain earns a living on the sales of the JBossCMP 2.0 documentation and as an independent consultant in Minneapolis, MN (USA).

The JBoss Group, LLC, headed by Marc Fleury, is composed of over 1000 developers worldwide who are working to deliver a full range of J2EE tools, making JBoss the premier Enterprise Java application server for the Java 2 Enterprise Edition platform.

JBoss is an open source, standards-compliant, J2EE application server implemented in 100% Pure Java. The JBoss/Server and complement of products are delivered under a public license. With 150,000 downloads per month, JBoss is the most downloaded J2EE based server in the industry.

Dedication

To Marleta for putting up with me all these years.

Acknowledgments

I would like to thank all of you who have posted bug reports. Without these reports, JBossCMP would have never become stable.

Chapter
0

0. Introduction to JBossCMP

What this Book Covers

JBossCMP is a powerful persistence engine compliant with the EJB 2.0 CMP 2.0 specification. This documentation explains how to configure JBossCMP for CMP 2.0. Specifically, it includes an introduction to each feature, along with its configuration, and a guide to specifying the database mapping of container managed data. Although the general reader may find this intellectually stimulating, the configuration of JBossCMP is not required to simply deploy and run an EJB 2.0 application; therefore, this documentation is intended primarily for those interested in using advanced features, specifying an exact database mapping, or tuning, all of which require further configuration.

This documentation assumes that the reader is familiar with Java, EJB and JBoss. It does not assume familiarity with JAWS, the JBoss CMP 1.1 persistence engine, although such familiarity may be helpful. For those without CMP experience, general CMP 2.0 coding and declaration are covered, although this documentation is by no means a complete introduction to CMP.

Organization

Each chapter of this documentation covers a specific feature of CMP 2.0. The first section of a chapter quickly introduces the feature, describes the java code required, and explains the declaration of the element in the ejb-jar.xml file. The remaining sections describe JBossCMP features and configuration. A short description of each chapter follows:

Chapter 1, Setup

> This chapter explains the setup of configuration files relevant to JBossCMP.

Chapter 2, Entities

> This chapter explains the configuration of entities with the exception of cmp-fields, cmr-fields, and queries, which are described in separate chapters.

Chapter 3, CMP-Fields

This chapter explains the configuration of cmp-fields and focuses on the new features such as eager/lazy loading, read-only fields, and dependent value classes.

Chapter 4, Container Managed Relationships

This chapter explains container managed relationships and the configuration of relationships for JBossCMP. The chapter focuses on the database relationship mapping.

Chapter 5, Queries

This chapter explains the declaration of queries for finder and ejbSelect methods, and how to override the EJB-QL to SQL mapping.

Chapter 6, Optimized Loading

This chapter explains the loading process and its configuration.

Appendix A, About The JBoss Group

This appendix contains information about The JBoss Group.

Appendix B, Defaults

This appendix explains the configuration of JBossCMP default options.

Appendix C, Datasource Customization

This appendix explains the configuration of a datasource.

Appendix D, Revision History

This appendix details the changes to JBossCMP and this document.

Chapter 1

1. Setup

JBossCMP is the default persistence manager for EJB 2.0 applications. Because JBossCMP is a core feature of JBoss 3.0, no action beyond the basic JBoss installation (see the **JBoss 3.0 Quick Start Guide**) is required to use CMP 2.0, but there are some details to note when creating a new EJB 2.0 application or when upgrading an EJB 1.1 application.

When JBoss deploys an EJB jar file, it uses the DOCTYPE of the ejb-jar.xml deployment descriptor to determine the version of the EJB jar. The correct DOCTYPE for EJB 2.0 follows:

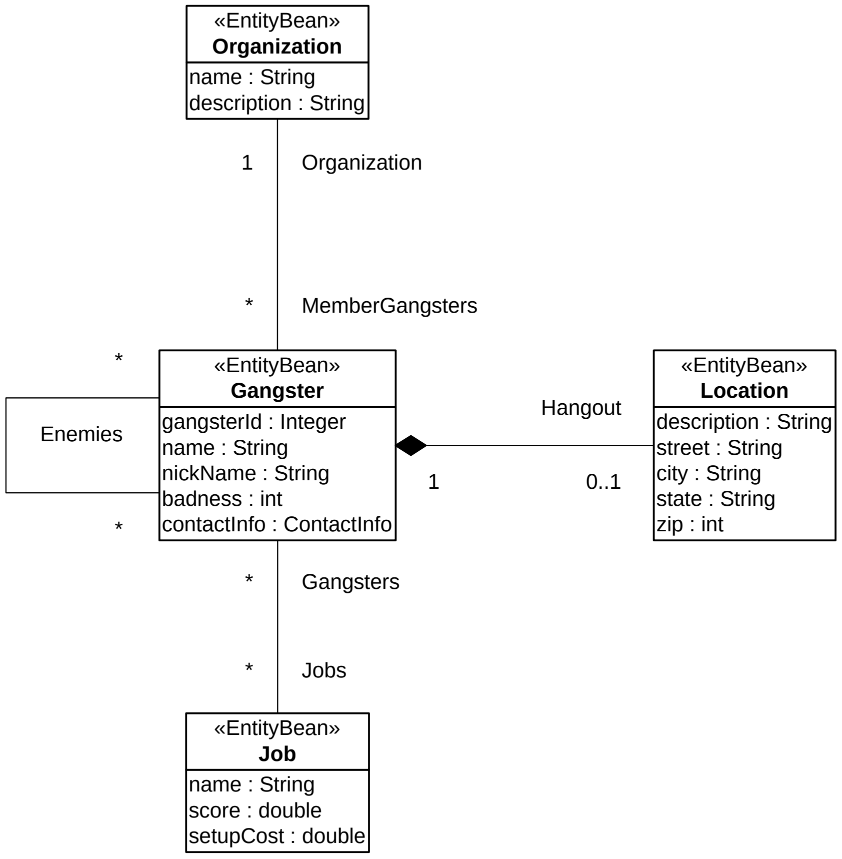
```
<!DOCTYPE ejb-jar PUBLIC
     "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
     "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
```

Listing 1-1, The EJB 2.0 DOCTYPE Declaration

If the public identifier of the DOCTYPE is "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" JBossCMP will use the "Standard CMP 2.x EntityBean" configuration in the standardjboss.xml file. If you have an application that uses a custom entity bean configuration, and you are upgrading to EJB 2.0, you must change the persistence-manager and add the new interceptors (see the "Standard CMP 2.x EntityBean" configuration in the standardjboss.xml file for details). No further configuration is necessary to deploy and run your EJB 2.0 application successfully.

Example Code

The full source code for all of the examples presented in this documentation is available as part of the downloaded package, and is based on the Crime Portal, which models criminal organizations. A diagram of the portions of the Criminal Portal data model used in the example code follows:



The example code requires jboss-3.0.0 and Jakarta Ant 1.4.1 or later. The setup of Ant is described in Chapter 0 of the JBoss 3.0 Quick Start Guide. The build.xml file in the example code relies on the JBOSS_HOME environment to find JBoss, so check that this variable is set before running Ant. Ant will give you the following output if JBOSS_HOME is not set:

```
$ ant
Buildfile: build.xml

cleandist:

prepare:
    [mkdir] Created dir: C:\work\jboss\cmp-example\output
    [mkdir] Created dir: C:\work\jboss\cmp-example\output\classes
    [mkdir] Created dir: C:\work\jboss\cmp-example\output\lib

compile:
    [javac] Compiling 22 source files to C:\work\jboss\cmp-example\output\classes

BUILD FAILED

c:\work\jboss\cmp-example\build.xml:55: C:\work\jboss\cmp-example\${env.JBOSS_HO
ME}\lib not found.
```

```

Total time: 0 seconds
```

To build the example code, simply execute Ant with no arguments. The output follows:

```
$ ant
Buildfile: build.xml

cleandist:

prepare:
    [mkdir] Created dir: C:\work\jboss\cmp-example\output
    [mkdir] Created dir: C:\work\jboss\cmp-example\output\classes
    [mkdir] Created dir: C:\work\jboss\cmp-example\output\lib

compile:
    [javac] Compiling 22 source files to C:\work\jboss\cmp-example\output\classes

build-ejb.jar:
      [jar] Building jar: C:\work\jboss\cmp-example\output\lib\gangster-cmp2.jar


deploy:
     [copy] Copying 1 file to c:\jboss-3.0.0\server\default\deploy

BUILD SUCCESSFUL

Total time: 3 seconds
```

This command builds and deploys the application in the JBoss server. When you start your JBoss server, or if it is already running, you should see the following deployment messages:

```
21:35:34,128 INFO  [MainDeployer] Starting deployment of package: file:/C:/jboss
-3.0.0/server/default/deploy/gangster-cmp2.jar
21:35:34,449 INFO  [EJBDeployer]
Bean   : GangsterEJB
Section: 10.6.2
Warning: CMP entity beans may not define the implementation of a finder.

21:35:34,519 INFO  [EJBDeployer]
Bean   : AutoNumberEJB
Method : public String ejbCreate(String)
Section: 10.6.4
Warning: The throws clause must define the javax.ejb.CreateException.

21:35:34,559 INFO  [EjbModule] Creating
21:35:34,910 INFO  [EjbModule] Deploying OrganizationEJB
21:35:34,930 INFO  [EjbModule] Deploying GangsterEJB
21:35:34,960 INFO  [EjbModule] Deploying JobEJB
21:35:34,980 INFO  [EjbModule] Deploying LocationEJB
21:35:35,000 INFO  [EjbModule] Deploying AutoNumberEJB
21:35:35,050 INFO  [EjbModule] Deploying EJBTestRunnerEJB
21:35:35,080 INFO  [EjbModule] Deploying ReadAheadEJB
```

```
21:35:35,641 INFO  [EjbModule] Created
21:35:35,641 INFO  [EjbModule] Starting
21:35:35,651 INFO  [LocationEJB] Created table 'LOCATION' successfully.
21:35:35,681 INFO  [OrganizationEJB] Created table 'ORGANIZATION' successfully.
21:35:35,741 INFO  [AutoNumberEJB] Created table 'AUTONUMBER' successfully.
21:35:35,751 INFO  [JobEJB] Created table 'JOB' successfully.
21:35:35,801 INFO  [GangsterEJB] Created table 'GANGSTER' successfully.
21:35:35,811 INFO  [GangsterEJB] Created table 'GANGSTER_ENEMIES' successfully.
21:35:35,841 INFO  [GangsterEJB] Created table 'GANGSTER_JOB' successfully.
21:35:35,961 INFO  [EjbModule] Started
21:35:35,961 INFO  [MainDeployer] Successfully completed deployment of package:
file:/C:/jboss-3.0.0/server/default/deploy/gangster-cmp2.jar
```

Ignore the two verifier messages. Normally verifier messages should not be ignored, but in this case, they are incorrect.[1] At this point you are ready to begin testing the application. There are three Ant targets central to testing: setup, test, and teardown. The setup target loads sample data into the database, the test target executes the unit test cases, and the teardown target removes the sample data from the database. The following shows the execution of the setup target:

```
$ ant setup
Buildfile: build.xml

prepare:

compile:

setup:
    [junit] .
    [junit] Time: 2.193
    [junit]
    [junit] OK (1 tests)
    [junit]

BUILD SUCCESSFUL

Total time: 4 seconds
```

The following shows execution of the test target:

```
$ ant test
Buildfile: build.xml

prepare:

compile:
```

[1] The first message "CMP entity beans may not define the implementation of a finder" is correct in warning that the EJB specification does not allow beans to implement a finder method; however, as JBossCMP does allow finders to be implemented using BMP custom finders, it can safely be ignored. The second message "The throws clause must define the javax.ejb.CreateException" is a bug in the JBoss auto-number sample code.

```
test:
    [junit] ...........
    [junit] Time: 1.522
    [junit]
    [junit] OK (11 tests)
    [junit]

BUILD SUCCESSFUL

Total time: 2 secondsTotal time: 2 seconds
```

The following shows the execution of the teardown target:

```
$ ant teardown
Buildfile: build.xml

prepare:

compile:

teardown:
    [junit] .
    [junit] Time: 0.801
    [junit]
    [junit] OK (1 tests)
    [junit]

BUILD SUCCESSFUL

Total time: 2 seconds
```

Read-ahead

A set of tests has been developed to demonstrate the optimized loading configurations presented in Chapter 6. Before the read-ahead tests can be run, the log level of JBossCMP must be increased. Logging in JBoss is handled by log4j, and log4j is controlled by the log4j.xml file in the server/default/conf directory. Two steps are necessary to setup the logging for the read-ahead tests: adjust the log levels of the org.jboss and org.jboss.ejb.plugins.cmp categories, and adjust the Console appender threshold. The following shows the changes to the log4j.xml file necessary to decrease the org.jboss category to INFO and the changes to increase the org.jboss.ejb.plugins.cmp category to DEBUG:

```
  <!-- ================ -->
  <!-- Limit categories -->
  <!-- ================ -->

  <!-- Limit JBoss categories to INFO -->
  <category name="org.jboss">
    <priority value="INFO"/>
  </category>
  <category name="org.jboss.ejb.plugins.cmp">
    <priority value="DEBUG"/>
  </category>
```

The following shows the changes to the log4j.xml file necessary to decrease the threshold of the Console appender:

```
  <!-- ============================== -->
  <!-- Append messages to the console -->
  <!-- ============================== -->

  <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
    <param name="Threshold" value="DEBUG"/>
    <param name="Target" value="System.out"/>

    <layout class="org.apache.log4j.PatternLayout">
      <!-- The default pattern: Date Priority [Category] Message\n -->
      <param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c{1}] %m%n"/>
    </layout>
  </appender>
```

Now that the logging is setup correctly, the read-ahead tests will display useful information (you may have to restart the JBoss server for it to recognize the changes to the log4j.xml file). The following shows the actual execution of the readahead Ant target:

```
$ ant readahead
Buildfile: build.xml

prepare:

compile:

readahead:
    [junit] .
    [junit] Time: 1.542
    [junit]
    [junit] OK (1 tests)
    [junit]

BUILD SUCCESSFUL

Total time: 2 seconds
```

When the readahead Ant target is executed, all of the SQL queries executed during the test are displayed in the JBoss server console. The important things to note when analyzing the output are the number of queries executed, the columns selected, and the number of rows loaded. The following shows the read-ahead none portion of the JBoss server console output from read-ahead test:

```
#########################################################
### read-ahead none
###
22:05:42,539 DEBUG [findAll_none] Executing SQL: SELECT t0_g.id FROM GANGSTER t0
_g ORDER BY t0_g.id ASC
22:05:42,559 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,559 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,579 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,609 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,619 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,639 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,649 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,669 DEBUG [GangsterEJB] Executing SQL: SELECT name, nick_name, badness,
 hangout, organization FROM GANGSTER WHERE (id=?)
22:05:42,689 INFO  [ReadAheadTest]
###
#########################################################
```

The following table contains a list of all of the Ant targets:

Table 1-1, Example Code Ant Targets

Target	Description
build	This target compiles the test code, builds the deployment ejb-jar, and copies the jar into the JBoss deployment directory causing the JBoss server to deploy the example application. This is the default target.
undeploy	This target deletes the application jar from the JBoss deployment directory causing the JBoss server to undeploy the application.
setup	This target loads the sample data into the database. This target can only be run after the application is deployed
test	This target executes the basic application test suite. This target can only be run after the sample data has been loaded into the database with the setup target.
test-gui	This target executes the basic application test suite using the JUnit Swing GUI. This target can only be run after the sample data has been loaded into the database with the setup target.

Target	Description
readahead	This target executes the read-ahead test suite. This target can only be run after the sample data has been loaded into the database with the setup target. Note this target is only useful after the log level has been configured as described in this chapter.
readahead-gui	This target executes the read-ahead test suite using the JUnit Swing GUI. This target can only be run after the sample data has been loaded into the database with the setup target. Note this target is only useful after the log level has been configured as described in this chapter.
teardown	This target removes the sample data that was loaded into the database with the setup target.
clean	This target deletes all of the class files and jars built for the test application.

Chapter

2

2. Entities

Entity Classes

Although several new features have been added, and there have been major changes to cmp-fields and finders, the basic entity bean structure has not changed much in CMP 2.0. A new feature of EJB 2.0 is the addition of local interfaces. A local interface is composed of two interfaces, the local interface and the local home interface.[2] These interfaces are conceptually the same thing as the remote interface and home interface (sometimes referred to as the remote home), except that local interfaces are only accessible within the same Java VM. This allows local interfaces to use pass-by-reference semantics, removing the overhead associated with serializing and deserializing every method parameter.[3] Local interfaces are not unique to CMP and are not discussed in this documentation. The simplified code for the Gangster entity follows:

```
// Gangster Local Home Interface
public interface GangsterHome extends EJBLocalHome {
   Gangster create(Integer id, String name, String nickName) throws CreateException;
   Gangster findByPrimaryKey(Integer id) throws FinderException;
}
```

Listing 2-1, Entity Local Home Interface

```
// Gangster Local Interface
public interface Gangster extends EJBLocalObject {
   Integer getGangsterId();
   String getName();
   String getNickName();
   void setNickName(String nickName);
}
```

Listing 2-2, Entity Local Interface

[2] The term local interface is used to refer to the EJBLocalObject alone, as well as to refer to the EJBLocalObject/ EJBLocalHome combination. Although this is confusing, it is the current usage of the term in the EJB community.

[3] Most J2EE servers, including JBoss, can optimize in VM calls over a remote interface by using pass-by-reference semantics.

```
// Gangster Implementation Class
public abstract class GangsterBean implements EntityBean {
   private EntityContext ctx;
   private Category log = Category.getInstance(getClass());

   public Integer ejbCreate(Integer id, String name, String nickName)
         throws CreateException {

      log.info("Creating Gangster " + id + " '" + nickName + "' "+ name);
      setGangsterId(id);
      setName(name);
      setNickName(nickName);
      return null;
   }

   public void ejbPostCreate(Integer id, String name, String nickName) { }

   // CMP field accessors -----------------------------------------------
   public abstract Integer getGangsterId();
   public abstract void setGangsterId(Integer gangsterId);

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

   public abstract String getNickName();
   public abstract void setNickName(String nickName);

   // EJB callbacks -----------------------------------------------------
   public void setEntityContext(EntityContext context) { ctx = context; }
   public void unsetEntityContext() { ctx = null; }
   public void ejbActivate() { }
   public void ejbPassivate() { }
   public void ejbRemove() { log.info("Removing " + getName()); }
   public void ejbStore() { }
   public void ejbLoad() {}
}
```
Listing 2-3, Entity Implementation Class

Entity Declaration

The declaration of an entity in the ejb-jar.xml file has not changed much in CMP 2.0. The declaration of the GangsterEJB follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC
   "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
   "http://java.sun.com/j2ee/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
   <enterprise-beans>
      <entity>
```

```
          <display-name>Gangster Entity Bean</display-name>
          <ejb-name>GangsterEJB</ejb-name>

          <local home>org.jboss.docs.cmp2.crimeportal.GangsterHome</local home>
          <local>org.jboss.docs.cmp2.crimeportal.Gangster</local>
          <ejb-class>org.jboss.docs.cmp2.crimeportal.GangsterBean</ejb-class>

          <persistence-type>Container</persistence-type>
          <prim-key-class>java.lang.Integer</prim-key-class>
          <reentrant>False</reentrant>
          <cmp-version>2.x</cmp-version>
          <abstract-schema-name>gangster</abstract-schema-name>

          <cmp-field><field-name>gangsterId</field-name></cmp-field>
          <cmp-field><field-name>name</field-name></cmp-field>
          <cmp-field><field-name>nickName</field-name></cmp-field>

          <primkey-field>gangsterId</primkey-field>
       </entity>
    </enterprise-beans>
 </ejb-jar>
```

Listing 2-4, The ejb-jar.xml Entity Declaration

The new local home and local elements are equivalent to the home and remote elements. The cmp-version element is new and can be either 1.x or the default 2.x. This element was added so 1.x and 2.x entities could be mixed in the same application. The abstract-schema-name element is also new and is used to identify this entity type in EJB-QL queries, which are discussed in Chapter 5.

Entity Mapping

The JBossCMP configuration for the entity is declared with an entity element in the jbosscmp-jdbc.xml file. This file is located in the META-INF directory of the ejb-jar file and contains all of the optional configuration information for JBossCMP. The entity elements are grouped together in the enterprise-beans element under the top level jbosscmp-jdbc element. An example entity configuration follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jbosscmp-jdbc PUBLIC
   "-//JBoss//DTD JBOSSCMP-JDBC 3.0//EN"
   "http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_3_0.dtd">

<jbosscmp-jdbc>
   <enterprise-beans>
         <ejb-name>GangsterEJB</ejb-name>
         <table-name>gangster</table-name>

         <!-- CMP Fields (see Chapter 3) -->
```

```
          <!-- Load Groups (see Chapter 6)-->

          <!-- Queries (see Chapter 5) -->

       </entity>
    </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 2-5, The jbosscmp-jdbc.xml Entity Mapping

In this case the DOCTYPE declaration is optional, but will reduce configuration errors. In addition, all of the elements are optional except for ejb-name, which is used to match the configuration to an entity declared in the ejb-jar.xml file. Unless noted otherwise, the default values come from the defaults section of the jbosscmp-jdbc.xml file, which is discussed in Appendix B. A detailed description of each entity element follows:

Table 2-1, entity Tags

Tag Name	Description	Required
ejb-name	This is the name of the EJB to which this configuration applies. This element must match an ejb-name of an entity in the ejb-jar.xml file.	Yes
datasource	This is the jndi-name used to look up the datasource. All database connections used by an entity or relation-table are obtained from the datasource. Having different datasources for entities is not recommended, as it vastly constrains the domain over which finders and ejbSelects can query.	No, default is java:/DefaultDS
datasource-mapping	This specifies the name of the type-mapping, which determines how Java types are mapped to SQL types, and how EJB-QL functions are mapped to database specific functions. Type-mapping is discussed in Appendix C.	No, default is Hypersonic SQL
create-table	If true, JBossCMP will attempt to create a table for the entity. When the application is deployed, JBossCMP checks if a table already exists before creating the table. If a table is found, it is logged, and the table is not created. This option is very useful during the early stages of development when the table structure changes often.	No, default is true
remove-table	If true, JBossCMP will attempt to drop the table for each entity and each relation-table mapped relationship. When the application is undeployed, JBossCMP will attempt to drop the table. This option is very useful during the early stages of development when the table structure changes often.	No, default is false

Tag Name	Description	Required
read-only	If true, the bean provider will not be allowed to change the value of any fields. A field that is read-only will not be stored in, or inserted into, the database. If a primary key field is read-only, the create method will throw a CreateException. If a set accessor is called on a read-only field, it throws an EJBException. Read-only fields are useful for fields that are filled in by database triggers, such as last update. The read-only option can be overridden on a per cmp-field basis, which is discussed in Chapter 3.	No, default is false
read-time-out	This is the amount of time in milliseconds that a read on a read-only field is valid. A value of 0 means that the value is always reloaded at the start of a transaction, and a value of -1 means that the value never times out. This option can also be overridden on a per cmp-field basis. If read-only is false, this value is ignored.	No, default is 300
row-locking	If true, JBossCMP will lock all rows loaded in a transaction. Most databases implement this by using the SELECT FOR UPDATE syntax when loading the entity, but the actual syntax is determined by the row-locking-template in the datasource-mapping used by this entity.	No, default is false
pk-constraint	If true, JBossCMP will add a primary key constraint when creating tables.	No, default is true
read-ahead	This controls caching of query results and cmr-fields for the entity. This option is discussed in Chapter 6.	No, see Chapter 6
list-cache-max	This specifies the number of read-lists that can be tracked by this entity. This option is discussed in Chapter 6.	No, default is 1000
table-name	This is the name of the table that will hold data for this entity. Each entity instance will be stored in one row of this table.	No, default is ejb-name

Chapter

3

3. CMP-Fields

CMP-Field Abstract Accessors

Although cmp-fields have not changed in CMP 2.0 with regards to functionality, they are no longer declared using fields in the bean implementation class. In CMP 2.0, cmp-fields are not directly accessible; rather each cmp-field is declared in the bean implementation class of the entity with a set of abstract accessor methods. Abstract accessors are similar to JavaBean property accessors, except no implementation is given. For example, the following listing declares the gangsterId, name, nickName, and badness cmp-fields in the gangster entity:

```
public abstract class GangsterBean implements EntityBean {
   public abstract Integer getGangsterId();
   public abstract void setGangsterId(Integer gangsterId);

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

   public abstract String getNickName();
   public abstract void setNickName(String param);

   public abstract int getBadness();
   public abstract void setBadness(int param);
}
```

Listing 3-1, cmp-field Abstract Accessor Declaration

Each cmp-field is required to have both a getter and a setter method, and each accessor method must be declared public abstract.

CMP-Field Declaration

The declaration of a cmp-field in the ejb-jar.xml file has not changed at all in EJB 2.0. For example, to declare the gangsterId, name, nickName and badness fields defined in *Listing 3-1* you would add the following to the ejb-jar.xml file:

```
<ejb-jar>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <cmp-field><field-name>gangsterId</field-name></cmp-field>
         <cmp-field><field-name>name</field-name></cmp-field>
         <cmp-field><field-name>nickName</field-name></cmp-field>
         <cmp-field><field-name>badness</field-name></cmp-field>
      </entity>
   </enterprise-beans>
</ejb-jar>
```

Listing 3-2, The ejb-jar.xml cmp-field Declaration

CMP-Field Column Mapping

The mapping of a cmp-field is declared in a cmp-field element within the entity. An example cmp-field mapping follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <table-name>gangster</table-name>
         <cmp-field>
            <field-name>gangsterId</field-name>
            <column-name>id</column-name>
         </cmp-field>
         <cmp-field>
            <field-name>name</field-name>
            <column-name>name</column-name>
            <not-null/>
         </cmp-field>
         <cmp-field>
            <field-name>nickName</field-name>
            <column-name>nick_name</column-name>
            <jdbc-type>VARCHAR</jdbc-type>
            <sql-type>VARCHAR(64)</sql-type>
         </cmp-field>
         <cmp-field>
            <field-name>badness</field-name>
            <column-name>badness</column-name>
         </cmp-field>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 3-3, The jbosscmp-jdbc.xml cmp-field Mapping

In the cmp-field element, you can control the name and datatype of the column. A detailed description of each element is shown in Table 3-1.

Table 3-1, cmp-field Tags

Tag Name	Description	Required
field-name	This is the name of the cmp-field that is being configured. It must match the name of a cmp-field declared for this entity in the ejb-jar.xml file.	Yes
column-name	This is the name of the column to which the cmp-field is mapped.	No, default is field-name
not-null	If this empty element is present, JBossCMP will add NOT NULL to the end of the column declaration when automatically creating the table for this entity.	No, default for primary key fields and primitives not-null
jdbc-type	This is the JDBC type that is used when setting parameters in a JDBC PreparedStatement or loading data from a JDBC ResultSet. The valid types are defined in java.sql.Types.	Only required if sql-type is specified, default is based on datasource-mapping
sql-type	This is the SQL type that is used in create table statements for this field. Valid sql-types are only limited by your database vendor.	Only required if jdbc-type is specified, default is based on datasource-mapping

Read-only Fields

Another benefit of abstract accessors for cmp-fields is the ability to have read-only fields. JAWS supported read-only with read-time-out for entities. The problem with CMP 1.x was the bean provider could always change the value of a field on a read-only entity, and there was nothing the container could do. With CMP 2.x, the container provides the implementation for the accessor, and therefore can throw an exception when the bean provider attempts to set the value of a read-only bean.

In JBossCMP this feature has been extended to the field level with the addition of the read-only and read-time-out elements to the cmp-field element. These elements work the same way as they do at the entity level. If a field is read-only, it will never be used in an INSERT or UPDATE statement. If a primary key field is read-only, the create method will throw a CreateException. If a set accessor is called for a read-only field, it throws an EJBException. Read-only fields are useful for fields that are filled in by database triggers, such as last update. A read-only cmp-field declaration example follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <cmp-field>
            <field-name>lastUpdated</field-name>
            <read-only>true</read-only>
            <read-time-out>1000</read-time-out>
         </cmp-field>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 3-4, The jbosscmp-jdbc.xml cmp-field read-only Declaration

Dependent Value Classes (DVCs)

A Dependent Value Class (DVC) is a fancy term used to identity any Java class that is the type of a cmp-field, other than the automatically recognized types. See section 10.3.3 of the **Enterprise JavaBeans Specification Version 2.0 Final Release** for further requirements. By default, a DVC is serialized, and the serialized form is stored in a single database column. Although not discussed here, there are several known issues with the long-term storage of classes in serialized form. JBossCMP supports the storage of the internal data of a DVC into one or more columns.

This is useful for supporting legacy JavaBeans and database structures. It is not uncommon to find a database with a highly flattened structure (e.g., a PURCHASE_ORDER table with the fields SHIP_LINE1, SHIP_LINE2, SHIP_CITY, etc. and an additional set of fields for the billing address). Other common database structures include telephone numbers with separate fields for area code, exchange, and extension, or a person's name spread across several fields. With a DVC, multiple columns can be mapped to one logical JavaBean. It is important to note that DVCs are not the same thing as Dependent Value Objects.[4]

JBossCMP requires that a DVC to be mapped must follow the JavaBeans naming specification for simple properties, and that each property to be stored in the database must have both a getter and a setter method.[5] Furthermore, the bean must be serializable and must have a no argument constructor. A property can be any simple type, an unmapped DVC or a mapped DVC, but

[4] Dependent Value Objects were added in Proposed Final Draft 1 of the EJB 2.0 Specification and subsequently replaced with local interfaces in Proposed Final Draft 2.

[5] The requirement that a DVC use the JavaBeans naming convention will be removed in a future release of JBossCMP. The current proposal is to allow the value to be retrieved from any no argument method and to be set with any single argument method or with a constructor.

cannot be an EJB.[6] An example declaration of a phone number DVC and contact information DVC follows:

```
<jbosscmp-jdbc>
   <dependent-value-classes>
      <dependent-value-class>
         <description>A phone number</description>
         <class>org.jboss.docs.cmp2.crimeportal.PhoneNumber</class>
         <property>
            <property-name>areaCode</property-name>
            <column-name>area_code</column-name>
         </property>
         <property>
            <property-name>exchange</property-name>
            <column-name>exchange</column-name>
         </property>
         <property>
            <property-name>extension</property-name>
            <column-name>extension</column-name>
         </property>
      </dependent-value-class>
      <dependent-value-class>
         <description>General contact info</description>
         <class>org.jboss.docs.cmp2.crimeportal.ContactInfo</class>
         <property>
            <property-name>cell</property-name>
            <column-name>cell</column-name>
         </property>
         <property>
            <property-name>pager</property-name>
            <column-name>pager</column-name>
         </property>
         <property>
            <property-name>email</property-name>
            <column-name>email</column-name>
            <jdbc-type>VARCHAR</jdbc-type>
            <sql-type>VARCHAR(128)</sql-type>
         </property>
      </dependent-value-class>
   </dependent-value-classes>
</jbosscmp-jdbc>
```

Listing 3-5, The jbosscmp-jdbc.xml Dependent Value Class Declaration

Each DVC is declared with a dependent-value-class element. A DVC is identified by the Java class type declared in the class element. Each property to be persisted is declared with a property element. This specification is based on the cmp-field element, so it should be self-explanatory.

[6] This restriction will also be removed in a future release. The current proposal involves storing the primary key fields in the case of a local entity and the entity handle in the case of a remote entity.

The dependent-value-classes section defines the internal structure and default mapping of the classes. When JBossCMP encounters a field that has an unknown type, it searches the list of registered DVCs, and if a DVC is found, it persists this field into a set of columns, otherwise the field is stored in serialized form in a single column. JBossCMP does not support inheritance of DVCs; therefore, this search is only based on the declared type of the field. A DVC can be constructed from other DVCs, so when JBossCMP runs into a DVC, it flattens the DVC tree structure into a set of columns. If JBossCMP finds a DVC circuit during startup, it will throw an EJBException. The default column name of a property is the column name of the base cmp-field followed by an underscore and then the property column name. If the property is a DVC, the process is repeated. For example, a cmp-field of type ContactInfo (see *Listing 3-5*) and named info will have the following columns:

```
info_cell_area_code
info_cell_exchange
info_cell_extension
info_pager_area_code
info_pager_exchange
info_pager_extension
info_email
```

Listing 3-6, Generated Column Names for ContactInfo Dependent Value Class

The automatically generated column names can quickly become excessively long and awkward. The default mappings of columns can be overridden in the entity element as follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <cmp-field>
            <field-name>contactInfo</field-name>
            <property>
               <property-name>cell.areaCode</property-name>
               <column-name>cell_area</column-name>
            </property>
            <property>
               <property-name>cell.exchange</property-name>
               <column-name>cell_exch</column-name>
            </property>
            <property>
               <property-name>cell.extension</property-name>
               <column-name>cell_ext</column-name>
            </property>
            <property>
               <property-name>pager.areaCode</property-name>
               <column-name>page_area</column-name>
            </property>
            <property>
               <property-name>pager.exchange</property-name>
               <column-name>page_exch</column-name>
            </property>
            <property>
               <property-name>pager.extension</property-name>
               <column-name>page_ext</column-name>
            </property>
            <property>
               <property-name>email</property-name>
               <column-name>email</column-name>
               <jdbc-type>VARCHAR</jdbc-type>
               <sql-type>VARCHAR(128)</sql-type>
            </property>
         </cmp-field>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 3-7, The jbosscmp-jdbc.xml cmp-field Dependent Value Class Property Override

As shown in *Listing 3-7*, when overriding property info for the entity, you need to refer to the property from a flat perspective as in cell.areaCode.

Chapter

4

4. Container Managed Relationships

Container Managed Relationships (CMRs) are a powerful new feature of CMP 2.0. Programmers have been creating relationships between entity objects since EJB 1.0 was introduced (not to mention since the introduction of databases), but before CMP 2.0 the programmer had to write a lot of code for each relationship in order to extract the primary key of the related entity and store it in a pseudo foreign key field. The simplest relationships were tedious to code, and complex relationships with referential integrity required many hours to code. With CMP 2.0 there is no need to code relationships by hand. The container can manage one-to-one, one-to-many and many-to-many relationships, with referential integrity. One restriction with CMRs is that they are only defined between local interfaces. This means that a relationship cannot be created between two entities in different virtual machines.[7]

There are two basic steps to create a container managed relationship: create the cmr-field abstract accessors and declare the relationship in the ejb-jar.xml file. The following two sections describe these steps.

CMR-Field Abstract Accessors

CMR-Field abstract accessors have the same signatures as cmp-fields, except that single-valued relationships must return the local interface of the related entity, and multi-valued relationships can only return a java.util.Collection (or java.util.Set) object. As with cmp-fields, at least one of the two entities in a relationship must have cmr-field abstract accessors. For example, to declare a one-to-many relationship between Organization and Gangster, first add the following to the OrganizationBean class:

```
public abstract class OrganizationBean implements EntityBean {
   public abstract Set getMemberGangsters();
   public abstract void setMemberGangsters(Set gangsters);
}
```

Listing 4-1, Collection Valued cmr-field Abstract Accessor Declaration

Second, add the following to the GangsterBean class:

[7] The EJB specification does not even allow for relationships between entities in different applications within the same VM.

```
public abstract class GangsterBean implements EntityBean {
   public abstract Organization getOrganization();
   public abstract void setOrganization(Organization org);
}
```

Listing 4-2, Single Valued cmr-field Abstract Accessor Declaration

Although in *Listing 4-1* and *Listing 4-2* each bean declared a cmr-field, only one of the two beans in a relationship must have a set of accessors. As with cmp-fields, a cmr-field is required to have both a getter and a setter method.

Relationship Declaration

The declaration of relationships in the ejb-jar.xml file is complicated and error prone. The XML used to declared relationships is as inconsistent as Visual Basic syntax. The best way to configure a relationship is to use a tool, such as **XDoclet**, or cut and paste a working relationship. The declaration of the Organization-Gangster relationship follows:

```
<ejb-jar>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Organization-Gangster</ejb-relation-name>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>org-has-gangsters</ejb-relationship-role-name>
            <multiplicity>One</multiplicity>
            <relationship-role-source>
               <ejb-name>OrganizationEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
               <cmr-field-name>memberGangsters</cmr-field-name>
               <cmr-field-type>java.util.Set</cmr-field-type>
            </cmr-field>
         </ejb-relationship-role>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-belongs-to-org</ejb-relationship-role-name>
            <multiplicity>Many</multiplicity>
            <cascade-delete/>
            <relationship-role-source>
               <ejb-name>GangsterEJB</ejb-name>
            </relationship-role-source>
            <cmr-field>
               <cmr-field-name>organization</cmr-field-name>
            </cmr-field>
         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</ejb-jar>
```

Listing 4-3, The ejb-jar.xml Relationship Declaration

As you can see, each relationship is declared with an ejb-relation element within the top level relationships[8] element, and each ejb-relation contains two ejb-relationship-role elements (one for each entity in the relationship). The ejb-relationship-role tags are described in the following table:

Table 4-1, ejb-relationship-role Tags

Tag Name	Description	Required
ejb-relationship-role-name	Used to identify the role and match the database mapping in the jbosscmp-jdbc.xml file. The name cannot be the same as the related role.	No
multiplicity	Must be "One" or "Many". Note, as with all XML elements, this element is case-sensitive.	Yes
cascade-delete	If this empty element is present, JBossCMP will delete the child entity when the parent entity is deleted. Cascade deletion is only allowed for a role where the other side of the relationship has a multiplicity of one.	No, default is to not cascade delete
relationship-role-source/ ejb-name	This is the entity that has the role.	Yes
cmr-field/ cmr-field-name	This is the name of the cmr-field if the entity has one.	Only required if entity has cmr-field abstract accessor
cmr-field/ cmr-field-type	This is the type of the cmr-field.[9] Must be java.util.Collection or java.util.Set.	Only required if cmr-field abstract accessor is collection valued

After adding the cmr-field abstract accessors and declaring the relationship, the relationship should be functional. For more information on relationships, see section 10.3 of the **Enterprise JavaBeans Specification Version 2.0 Final Release**. The next section discusses the database mapping of the relationship.

[8] This is the first place where the specification is inconsistent. It would be much easier if the specification used the following tags: relationships, relationship, and relationship-name.

[9] This is another place where the spec goes awry, as the tag is completely unnecessary. The cmr-field-type is readily accessible in the cmr-field get accessor method return type, and return types cannot be overridden.

Relationship Mapping

Relationships can be mapped using either a foreign key or a separate relation-table. One-to-one and one-to-many relationships use the foreign key mapping style by default, and many-to-many relationships use only the relation-table mapping style. The mapping of a relationship is declared in the relationships section of the jbosscmp-jdbc.xml file. Relationships are identified by the ejb-relation-name from the ejb-jar.xml file. The basic template of the relationship mapping declaration for Organization-Gangster follows:

```
<jbosscmp-jdbc>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Organization-Gangster</ejb-relation-name>

         <!--
          | Mapping style declaration
          | <foreign-key> or <relation-table>
          -->

         <read-only>false</read-only>
         <read-time-out>300</read-time-out>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>org-has-gangsters</ejb-relationship-role-name>

            <fk-constraint>true</fk-constraint>

            <key-fields>
               <!-- Organization primary key field mappings -->
            </key-fields>

            <read-ahead><strategy>on-load</strategy></read-ahead>

         </ejb-relationship-role>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-belongs-to-org</ejb-relationship-role-name>
            <fk-constraint>true</fk-constraint>

            <key-fields>
               <!— Gangster primary key field mappings -->
            </key-fields>

            <read-ahead><strategy>on-load</strategy></read-ahead>

         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</jbosscmp-jdbc>
```

Listing 4-4, The jbosscmp-jdbc.xml Relationship Mapping Template

After the ejb-relation-name of the relationship being mapped is declared, the mapping style is declared using a foreign-key-mapping element or a relation-table-mapping element, both of which are discussed in the next two sections. The read-only and read-time-out elements have the same semantics they did in the entity element (see Chapter 2). The ejb-relationship-role elements are optional, but if one is declared, the other must also be declared. A detailed description of the elements contained in the ejb-relationship-role element follows:

Table 4-2, ejb-relation Tags

Tag Name	Description	Required
ejb-relation-name	This is the name of the relationship that is being configured. This element must match the name of a relationship declared in the ejb-jar.xml file.	Yes
read-only	If true, the bean provider will not be allowed to change the value of this relationship. A relationship that is read-only will not be stored in, or inserted into, the database. If a set accessor is called on a read-only relationship, it throws an EJBException.	No, default is false
read-time-out	This is the amount of time in milliseconds that a read on a read-only relationship is valid. A value of 0 means that the value is always reloaded at the start of a transaction, and a value of -1 means that the value never times out. If read-only is false, this value is ignored.	No, default is 300

The ejb-relation element must contain either a foreign-key-mapping element or a relation-table-mapping element, which are described in the foreign key mapping and relation-table mapping sections respectively. This element may also contain a pair of ejb-relationship-role elements as described in the following section.

Relationship Role Mapping

Each of the two ejb-relationship-role elements contains mapping information specific to an entity in the relationship. A detailed description of the main elements follows:

Table 4-3, ejb-relationship-role Tags

Tag Name	Description	Required
ejb-relationship-role-name	This is the name of the role to which this configuration applies. This element must match the name of one of the roles declared for this query in the ejb-jar.xml file.	Yes

Tag Name	Description	Required
fk-constraint	If true, JBossCMP will add a foreign key constraint to the tables. JBossCMP will only add the constraint if both the primary table and the related table were created by JBossCMP during deployment.	No, default is false
key-fields	This specifies the mapping of the primary key fields of the current entity. This element is only necessary if exact field mapping is desired. Otherwise, the key-fields element must[10] contain a key-field element for each primary key field of the current entity. The details of this element are described below.	No, default depends on mapping type
read-ahead	This controls the caching of this relationship. This option is discussed in Chapter 6.	No, see Chapter 6

As noted in *Table 4-3* the key-fields element contains a key-field for each primary key field of the current entity. The key-field element uses the same syntax as the cmp-field element of the entity, except that key-field does not support the not-null option. Key-fields of a relation-table are automatically not null, because they are the primary key of the table. On the other hand, foreign key fields must always be nullable.[11] A detailed description of the elements contained in the key-field element follows:

Table 4-4, key-field Tags

Tag Name	Description	Required
field-name	This identifies the field to which this mapping applies. This name must match a primary key field of the current entity.	Yes

[10] Note that with foreign key mapping this element can be empty; this means that there will be not be a foreign key for the current entity. This is required for the many side of a one-to-many relationship, such a Gangster in the Organization-Gangster example.

[11] The current implementation of JBossCMP inserts a row into the database for a new entity between ejbCreate and ejbPostCreate. Since the EJB specification does not allow a relationship to be modified until ejbPostCreate, a foreign key will be initially set to null. There is a similar problem with removal. This limitation will be removed in a future release.

Tag Name	Description	Required
column-name	Specifies the column name in which this primary key field will be stored. If this is relationship uses foreign-key-mapping, this column will be added to the table for the related entity. If this relationship uses relation-table-mapping, this column is added to the relation-table. This element is not allowed for mapped dependent value class; instead use the property element described in Chapter 3.	No, default depends on mapping type
jdbc-type	This is the JDBC type that is used when setting parameters in a JDBC PreparedStatement or loading data from a JDBC ResultSet. The valid types are defined in java.sql.Types.	Only required if sql-type is specified, default is based on datasource-mapping
sql-type	This is the SQL type that is used in create table statements for this field. Valid sql-types are only limited by your database vendor.	Only required if jdbc-type is specified, default is based on datasource-mapping

Foreign Key Mapping

Foreign key mapping is the most common mapping style for one-to-one and one-to-many relationships, but is not allowed for many-to many relationships. The foreign key mapping element is simply declared by adding an empty foreign key-mapping element to the ejb-relation element.

As noted in the previous section, with a foreign key mapping the key-fields declared in the ejb-relationship-role are added to the table of the related entity. If the key-fields element is empty, a foreign key will not be created for the entity. In a one-to-many relationship, the many side (Gangster in the example) must have an empty key-fields element, and the one side (Organization in the example) must have a key-fields mapping. In one-to-one relationships, one or both roles can have foreign keys.

The foreign key mapping is not dependent on the direction of the relationship. This means that in a one-to-one unidirectional relationship (only one side has an accessor) one or both roles can still have foreign keys.

The complete foreign key mapping for the Organization-Gangster relationship follows:

```
<jbosscmp-jdbc>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Organization-Gangster</ejb-relation-name>
         <foreign-key-mapping/>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>org-has-gangsters</ejb-relationship-role-name>
            <key-fields>
               <key-field>
                  <field-name>name</field-name>
                  <column-name>organization</column-name>
               </key-field>
            </key-fields>
         </ejb-relationship-role>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-belongs-to-org</ejb-relationship-role-name>
            <key-fields/>
         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</jbosscmp-jdbc>
```

Listing 4-5, The jbosscmp-jdbc.xml Foreign Key Mapping

Relation-table Mapping

Relation-table mapping is less common for one-to-one and one-to-many relationships, but is the only mapping style allowed for many-to-many relationships. The relation-table-mapping for the Gangster-Job relationship follows:

```
<jbosscmp-jdbc>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Gangster-Jobs</ejb-relation-name>
         <relation-table-mapping>
            <table-name>gangster_job</table-name>
         </relation-table-mapping>

         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-has-jobs</ejb-relationship-role-name>
            <key-fields>
               <key-field>
                  <field-name>gangsterId</field-name>
                  <column-name>gangster</column-name>
               </key-field>
            </key-fields>
         </ejb-relationship-role>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>job-has-gangsters</ejb-relationship-role-name>
            <key-fields>
               <key-field>
                  <field-name>name</field-name>
                  <column-name>job</column-name>
               </key-field>
            </key-fields>
         </ejb-relationship-role>
      </ejb-relation>
   </relationships>
</jbosscmp-jdbc>
```

Listing 4-6, The jbosscmp-jdbc.xml Relation-table Mapping

The relation-table-mapping element contains a subset of the options available in the entity element. A detailed description of these elements is reproduced here for convenience:

Table 4-5, relation-table-mapping Tags

Tag Name	Description	Required
table-name	This is the name of the table that will hold data for this relationship.	No, default is based on entity and cmr-field names
datasource	This is the jndi-name used to look up the datasource. All database connections are obtained from the datasource. Having different datasources for entities is not recommended, as it vastly constrains the domain over which finders and ejbSelects can query.	No, default is java:/DefaultDS
datasource-mapping	This specifies the name of the type-mapping, which determines how Java types are mapped to SQL types, and how EJB-QL functions are mapped to database specific functions. Type-mapping is discussed in Appendix C.	No, default is Hypersonic SQL
create-table	If true, JBossCMP will attempt to create a table for the relationship. When the application is deployed, JBossCMP checks if a table already exists before creating the table. If a table is found, it is logged, and the table is not created. This option is very useful during the early stages of development when the table structure changes often.	No, default is true

Tag Name	Description	Required
remove-table	If true, JBossCMP will attempt to drop the relation-table when the application is undeployed. This option is very useful during the early stages of development when the table structure changes often.	No, default is false
row-locking	If true, JBossCMP will lock all rows loaded in a transaction. Most databases implement this by using the SELECT FOR UPDATE syntax when loading the entity, but the actual syntax is determined by the row-locking-template in the datasource-mapping used by this entity.	No, default is false
pk-constraint	If true, JBossCMP will add a primary key constraint when creating tables.	No, default is true

Chapter
5

5. Queries

Another powerful new feature of CMP 2.0 is the introduction of the EJB Query Language (EJB-QL) and ejbSelect methods. In CMP 1.1, every EJB container had a different way to specify finders, and this was a serious threat to J2EE portability. In CMP 2.0, EJB-QL was created to specify finders and ejbSelect methods in a platform independent way. The ejbSelect method is designed to provide private query statements to an entity implementation. Unlike finders, which are restricted to only return entities of the same type as the home interface on which they are defined, ejbSelect methods can return any entity type or just one field of the entity.

EJB-QL is beyond the scope of this documentation, so only the basic method coding and query declaration will be covered here. For more information, see Chapter 11 of the **Enterprise JavaBeans Specification Version 2.0 Final Release** or one of the many excellent articles written on CMP 2.0.

Finder and ejbSelect Declaration

The declaration of finders has not changed in CMP 2.0. Finders are still declared in the home interface (local or remote) of the entity. Finders defined on the local home interface do not throw a RemoteException. The following code declares the findBadDudes_ejbql[12] finder on the GangsterHome interface:

```
public interface GangsterHome extends EJBLocalHome {
   Collection findBadDudes_ejbql(int badness) throws FinderException;
}
```

Listing 5-1, Finder Declaration

The ejbSelect methods are declared in the entity implementation class, and must be public abstract just like cmp-field and cmr-field abstract accessors. Select methods must be declared to throw a FinderException, but not a RemoteException. The following code declares an ejbSelect method:

[12] Ignore the "ejbql" suffix; it is not required. Later this query will be implemented using JBossQL and DeclaredSQL, and the suffix is used to separate the different query specifications in the jbosscmp-jdbc.xml file.

```
public abstract class GangsterBean implements EntityBean {
   public abstract Set ejbSelectBoss_ejbql(String name) throws FinderException;
}
```

Listing 5-2, ejbSelect Declaration

EJB-QL Declaration

The EJB 2.0 specification requires that every ejbSelect or finder method (except findByPrimaryKey) have an EJB-QL query defined in the ejb-jar.xml file.[13] The EJB-QL query is declared in a query element, which is contained in the entity element. The following is the declaration for the queries defined in *Listing 5-1* and *Listing 5-2*:

```
<ejb-jar>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findBadDudes_ejbql</method-name>
               <method-params><method-param>int</method-param></method-params>
            </query-method>
            <ejb-ql><![CDATA[
               SELECT OBJECT(g)
               FROM gangster g
               WHERE g.badness > ?1
            ]]></ejb-ql>
         </query>
         <query>
            <query-method>
               <method-name>ejbSelectBoss_ejbql</method-name>
               <method-params>
                  <method-param>java.lang.String</method-param>
               </method-params>
            </query-method>
            <ejb-ql><![CDATA[
               SELECT DISTINCT underling.organization.theBoss
               FROM gangster underling
               WHERE underling.name = ?1 OR underling.nickName = ?1
            ]]></ejb-ql>
         </query>
      </entity>
   </enterprise-beans>
</ejb-jar>
```

Listing 5-3, The ejb-jar.xml Query Declaration

[13] Currently this is not enforced by JBossCMP, but a future release will enforce this by throwing an exception during deployment.

EJB-QL is similar to SQL but has some surprising differences. The following are some important things to note about EJB-QL:

- EJB-QL is a typed language, meaning that it only allows comparison of like types (i.e., strings can only be compared with strings).
- In an equals comparison a variable (single valued path) must be on the left hand side. Some examples follow: [14]

```
g.hangout.state = 'CA'                  Legal
'CA' = g.shippingAddress.state          NOT Legal
'CA' = 'CA'                             NOT Legal
(r.amountPaid * .01) > 300              NOT Legal
r.amountPaid > (300 / .01)              Legal
```

- Parameters use a base 1 index like java.sql.PreparedStatement.
- Parameters are only allowed on the right hand side of a comparison. For example:

```
gangster.hangout.state = ?1             Legal
?1 = gangster.hangout.state             NOT Legal
```

Overriding the EJB-QL to SQL Mapping

The EJB-QL to SQL mapping can be overridden in the jbosscmp-jdbc.xml file. The finder or ejbSelect is still required to have an EJB-QL declaration in the ejb-jar.xml file, but the ejb-ql element can be left empty. Currently the SQL can be overridden with JBossQL, DynamicQL, DeclaredSQL or a BMP style custom ejbFind method. All EJB-QL overrides are non-standard extensions to the EJB 2.0 specification, so use of these extensions will limit portability of your application. All of the EJB-QL overrides, except for BMP custom finders, use the following declaration template:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findBadDudes</method-name>
               <method-params><method-param>int</method-param></method-params>
```

[14] The example "(r.amountPaid * .01) > 300" is presented on page 244 of "Enterprise JavaBeans 3rd Edition" by Richard Monson-Haefel to demonstrate the use of arithmetic operators in a WHERE clause, and is included here to highlight the fact that it is not legal EJB-QL syntax.

```
            </query-method>
            <!—-
             | ejb-ql override here
             | <jboss-ql>, <dynamic-ql>, or <declared-sql>
            -->
         </query>
      </entity>
   </enterprise-beans>
</ejb-jar>
```

Listing 5-4, The jbosscmp-jdbc.xml EJB-QL Override Template

JBossQL

JBossQL is a superset of EJB-QL that is designed to address some of the inadequacies of EJB-QL. In addition to a more flexible syntax, new functions, key words, and clauses have been added to JBossQL. At the time of this writing, JBossQL includes support for an ORDER BY clause, parameters in the IN and LIKE operators, and the UCASE and LCASE functions. The modifications to the EJB-QL BNF follow:

```
JBossQL := select_clause from_clause [where_clause] [order_by_clause]

order_by_clause := ORDER BY order_by_path_expression (, order_by_path_expression)*

order_by_path_expression :=
      ( numeric_valued_path | string_valued_path | datetime_valued_path )
      [ASC | DESC]

in_expression ::=
      single_valued_path_expression [NOT ]IN
      ( (string_literal | string_valued_parameter)
        [, (string_literal | string_valued_parameter) ]* )

like_expression ::=
      single_valued_path_expression [NOT ]LIKE
      (pattern_value | string_valued_parameter)
      [ESCAPE (escape-character | string_valued_parameter) ]

functions_returning_strings ::=
      CONCAT (string_expression, string_expression) |
      SUBSTRING (string_expression, arithmetic_expression, arithmetic_expression)
      UCASE (string_expression) |
      LCASE (string_expression)
```

Listing 5-5, JBossQL Expanded BNF

JBossQL is declared in the jbosscmp-jdbc.xml file with a jboss-ql element containing the JBossQL query. See *Listing 5-6* below for an example JBossQL declaration and *Listing 5-7* for the generated SQL:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findBadDudes_jbossql</method-name>
               <method-params><method-param>int</method-param></method-params>
            </query-method>
            <jboss-ql><![CDATA[
               SELECT OBJECT(g)
               FROM gangster g
               WHERE g.badness > ?1
               ORDER BY g.badness DESC
            ]]></jboss-ql>
         </query>
      </entity>
   </enterprise-beans>
</ejb-jar>
```

Listing 5-6, The jbosscmp-jdbc.xml JBossQL Override

```
SELECT t0_g.id
FROM gangster t0_g
WHERE t0_g.badness > ?
ORDER BY t0_g.badness DESC
```

Listing 5-7, JBossQL SQL Mapping

DynamicQL

DynamicQL allows the runtime generation and execution of JBossQL queries. A DynamicQL query method is an abstract method that takes the JBossQL query and the query arguments as parameters. JBossCMP compiles the JBossQL and executes the generated SQL. The following generates a JBossQL query that selects all the gangsters that have a hangout in any state in the states set:

```
public abstract class GangsterBean implements EntityBean {

   public abstract Set ejbSelectGeneric(String jbossQl, Object[] arguments)
         throws FinderException;

   public Set ejbHomeSelectInStates(Set states) throws FinderException {
      // generate JBossQL query
      StringBuffer jbossQl = new StringBuffer();
      jbossQl.append("SELECT OBJECT(g) ");
      jbossQl.append("FROM gangster g ");
      jbossQl.append("WHERE g.hangout.state IN (");
      for(int i = 0; i < states.size(); i++) {
         if(i > 0) {
            jbossQl.append(", ");
         }
         jbossQl.append("?").append(i+1);
      }
      jbossQl.append(") ORDER BY g.name");

      // pack arguments into an Object[]
      Object[] args = states.toArray(new Object[states.size()]);

      // call dynamic-ql query
      return ejbSelectGeneric(jbossQl.toString(), args);
   }
}
```

Listing 5-8, DynamicQL Example Code

The DynamicQL ejbSelect method may have any valid ejbSelect method name, but the method must always take a String and Object array as parameters. DynamicQL is declared in the jbosscmp-jdbc.xml file with an empty dynamic-ql element. The following is the declaration for the query defined in *Listing 5-8*:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>ejbSelectGeneric</method-name>
               <method-params>
                  <method-param>java.lang.String</method-param>
                  <method-param>java.lang.Object[]</method-param>
               </method-params>
            </query-method>
            <dynamic-ql/>
         </query>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 5-9, The jbosscmp-jdbc.xml DynamicQL Override

DeclaredSQL

DeclaredSQL is based on the JAWS finder declaration, but has been updated for CMP 2.0. Commonly this declaration is used to limit a query with a WHERE clause that cannot be represented in EJB-QL or JBossQL. See *Listing 5-10* below for an example DeclaredSQL declaration and *Listing 5-11* for the generated SQL:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findBadDudes_declaredsql</method-name>
               <method-params><method-param>int</method-param></method-params>
            </query-method>
            <declared-sql>
               <where><![CDATA[ badness > {0} ]]></where>
```

```
                <order><![CDATA[ badness DESC ]]></order>
             </declared-sql>
          </query>
       </entity>
    </enterprise-beans>
</ejb-jar>
```

Listing 5-10, The jbosscmp-jdbc.xml DeclaredSQL Override

```
SELECT id
FROM gangster
WHERE badness > ?
ORDER BY badness DESC
```

Listing 5-11, DeclaredSQL SQL Mapping

As you can see, JBossCMP generates the SELECT and FROM clauses necessary to select the primary key for this entity. If desired an additional FROM clause can be specified that is appended to the end of the automatically generated FROM clause. See *Listing 5-12* below for an example DeclaredSQL declaration with an additional FROM clause and *Listing 5-13* for the generated SQL:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>ejbSelectBoss_declaredsql</method-name>
               <method-params>
                  <method-param>java.lang.String</method-param>
               </method-params>
            </query-method>
            <declared-sql>
               <select>
                  <distinct/>
                  <ejb-name>GangsterEJB</ejb-name>
                  <alias>boss</alias>
               </select>
               <from><![CDATA[ , gangster g, organization o ]]></from>
               <where><![CDATA[
                  (LCASE(g.name) = {0} OR LCASE(g.nick_name) = {0}) AND
                  g.organization = o.name AND o.the_boss = boss.id
               ]]></where>
            </declared-sql>
         </query>
      </entity>
   </enterprise-beans>
</ejb-jar>
```

Listing 5-12, The jbosscmp-jdbc.xml DeclaredSQL Override With From Clause

```
SELECT DISTINCT boss.id
FROM gangster boss, gangster g, organization o
WHERE (LCASE(g.name) = ? OR LCASE(g.nick_name) = ?) AND
      g.organization = o.name AND o.the_boss = boss.id
```

Listing 5-13, The jbosscmp-jdbc.xml DeclaredSQL With From Clause SQL Mapping

Notice that the FROM clause starts with a comma. This is because the container appends the declared FROM clause to the end of the generated FROM clause. It is also possible for the FROM clause to start with a SQL JOIN statement. Since this is an ejbSelect method, it must have a select element to declare the entity that will be selected. Note that an alias is also declared for the query. If an alias is not declared, the table-name is used as the alias, resulting in a SELECT clause with the table_name.field_name style column declarations. Not all database vendors support the table_name.field_name syntax, so the declaration of an alias is preferred. *Listing 5-13* also used the optional empty distinct element, which causes the SELECT clause to use the SELECT DISTINCT declaration. The DeclaredSQL declaration can also be used in ejbSelect methods to select a cmp-field. The following example selects all of the zip codes in which an Organization operates:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>OrganizationEJB</ejb-name>
         <query>
            <query-method>
               <method-name>ejbSelectOperatingZipCodes_declaredsql</method-name>
               <method-params>
                  <method-param>java.lang.String</method-param>
               </method-params>
            </query-method>
            <declared-sql>
               <select>
                  <distinct/>
                  <ejb-name>LocationEJB</ejb-name>
                  <field-name>zipCode</field-name>
                  <alias>hangout</alias>
               </select>
               <from>, organization o, gangster g</from>
               <where>
                  LCASE(o.name) = {0} AND o.name = g.organization AND
                  g.hangout = hangout.id
               </where>
               <order>hangout.zip</order>
            </declared-sql>
         </query>
      </entity>
   </enterprise-beans>
</ejb-jar>
```

Listing 5-14, The jbosscmp-jdbc.xml DeclaredSQL ejbSelect Override

```
SELECT DISTINCT hangout.zip
FROM location hangout, organization o, gangster g
WHERE LCASE(o.name) = ? AND o.name = g.organization AND g.hangout = hangout.id
ORDER BY hangout.zip
```

Listing 5-15, The jbosscmp-jdbc.xml DeclaredSQL ejbSelect SQL Mapping

The following table describes each element of the select clause:

Table 5-1, select Tags

Tag Name	Description	Required
distinct	If this empty element is present, JBossCMP will add the DISTINCT keyword to the generated SELECT clause.	No, default is to use DISTINCT if method returns a java.util.Set
alias	This specifies the alias that will be used for the main select table.	No, default is ejb-name
ejb-name	This is the ejb-name of the entity that will be selected.	No, required if ejbSelect method
field-name	This is the name of the cmp-field that will be selected from the specified entity.	No, default is to select entire entity

Parameters

JBossCMP DeclaredSQL uses a completely new parameter handling system, which supports entity and DVC parameters. Parameters are enclosed in curly brackets and use a base zero index, which is different from the base one EJB-QL parameters. There are three categories of parameters: simple, DVC, and entity:

- A **simple** parameter can be of any type except for a known (mapped) DVC or an entity. A simple parameter only contains the argument number, such as {0}. When a simple parameter is set, the JDBC type used to set the parameter is determined by the datasource-mapping for the entity. An unknown DVC is serialized and then set as a parameter. Note that most databases do not support the use of a BLOB value in a WHERE clause.

- A **DVC** parameter can be any known (mapped) DVC. A DVC parameter must be dereferenced down to a simple property (one that is not another DVC). For example, if we had a property of type ContactInfo (as declared in Chapter 3), valid parameter declarations would be {0.email} and {0.cell.areaCode} but not {0.cell}. The JDBC type used to set a parameter is based on the class type of the property and the datasource-mapping of the entity. The JDBC type used to set the parameter is the JDBC type that is declared for that property in the dependent-value-class element.

- An **entity** parameter can be any entity in the application. An entity parameter must be dereferenced down to a simple primary key field or simple property of a DVC primary key field. For example, if we had a parameter of type Gangster, a valid parameter declaration would be {0.gangsterId}. If we had some entity with a primary key field named info of type ContactInfo (as declared in Chapter 3), a valid parameter declaration would be {0.info.cell.areaCode}. Only fields that are members of the primary key of the entity can be dereferenced (this restriction may be removed in later versions). The JDBC type used to set the parameter is the JDBC type that is declared for that field in the entity declaration.

BMP Custom Finders

JBossCMP continues the tradition of JAWS in supporting bean managed persistence custom finders. If a custom finder matches a finder declared in the home or local home interface, JBossCMP will always call the custom finder over any other implementation declared in the ejb-jar.xml or jbosscmp-jdbc.xml files. The following simple example finds the entities by a collection of primary keys:[15]

```
public abstract class GangsterBean implements EntityBean {
   public Collection ejbFindByPrimaryKeys(Collection keys) {
      return keys;
   }
}
```

Listing 5-16, Custom Finder Example Code

[15] This is a very useful finder because it quickly coverts primary keys into real Entity objects without contacting the database. One drawback is that it can create an Entity object with a primary key that does not exist in the database. If any method is invoked on the bad Entity, a NoSuchEntityException will be thrown.

Chapter

6

6. Optimized Loading

The goal of optimized loading is to load the smallest amount of data required to complete a transaction in the least number of queries. The tuning of JBossCMP depends on a detailed knowledge of the loading process. This chapter describes the internals of the JBossCMP loading process and its configuration. Tuning of the loading process really requires a holistic understanding of the loading system, so this chapter may have to be read more than once.

Loading Scenario

The easiest way to investigate the loading process is to look at a usage scenario. The most common scenario is to locate a collection of entities and iterate over the results performing some operation. The following example generates an html table containing all of the gangsters:

```
public String createGangsterHtmlTable_none() throws FinderException {
   StringBuffer table = new StringBuffer();
   table.append("<table>");

   Collection gangsters = gangsterHome.findAll_none();
   for(Iterator iter = gangsters.iterator(); iter.hasNext(); ) {
      Gangster gangster = (Gangster)iter.next();
      table.append("<tr>");
      table.append("<td>").append(gangster.getName()).append("</td>");
      table.append("<td>").append(gangster.getNickName()).append("</td>");
      table.append("<td>").append(gangster.getBadness()).append("</td>");
      table.append("</tr>");
   }

   table.append("</table>");
   return table.toString();
}
```

Listing 6-1, Loading Scenario Example Code

Assume this code is called within a single transaction and all optimized loading has been disabled. At Arrow 1, JBossCMP will execute the following query:

```
SELECT t0_g.id
FROM gangster t0_g
```

```
ORDER BY t0_g.id ASC
```

Listing 6-2, Unoptimized findAll Query

Then at Arrow 2, in order to load the eight Gangsters in the sample database, JBossCMP executes the following eight queries:

```
SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=0)

SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=1)

SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=2)

SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=3)

SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=4)

SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=5)

SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=6)

SELECT name, nick_name, badness, hangout, organization
FROM gangster WHERE (id=7)
```

Listing 6-3, Unoptimized Load Queries

There are two problems with this scenario. First, an excessive number of queries are executed because JBossCMP executes one query for findAll and one query for each element found; this is called the "n+1" problem[16] and is addressed with the read-ahead strategies described in the following sections. Second, values of unused fields are loaded because JBossCMP loads the

[16] The reason for this behavior has to do with the handling of query results inside the JBoss container. Although it appears that the actual entity beans selected are returned when a query is executed, JBoss really only returns the primary keys of the matching entities, and does not load the entity until a method is invoked on it.

hangout and organization fields,[17] which are never accessed. Configuration of eager loading is described in the Eager-loading Process section of this chapter. The following table shows the execution of the queries:

Table 6-1, Unoptimized Query Execution

id	name	nick_name	badness	hangout	organization
0	Yojimbo	Bodyguard	7	0	Yakuza
1	Takeshi	Master	10	1	Yakuza
2	Yuriko	Four finger	4	2	Yakuza
3	Chow	Killer	9	3	Triads
4	Shogi	Lightning	8	4	Triads
5	Valentino	Pizza-Face	4	5	Mafia
6	Toni	Toothless	2	6	Mafia
7	Corleone	Godfather	6	7	Mafia

Load Groups

The configuration and optimization of the loading system begins with the declaration of named load groups in the entity. A load group contains the names of cmp-fields and cmr-fields with a foreign key (e.g., Gangster in the Organization-Gangster example) that will be loaded in a single operation. An example configuration follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <load-groups>
            <load-group>
               <load-group-name>basic</load-group-name>
               <field-name>name</field-name>
               <field-name>nickName</field-name>
               <field-name>badness</field-name>
```

[17] Normally JBossCMP would also load the contactInfo field, but for the sake of readability, it has been disabled in this example because contact info maps to seven columns. The actual configuration used to disable the default loading of the contactInfo field is presented in *Listing 6-12*.

```
            </load-group>
            <load-group>
               <load-group-name>contact info</load-group-name>
               <field-name>nickName</field-name>
               <field-name>contactInfo</field-name>
               <field-name>hangout</field-name>
            </load-group>
         </load-groups>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 6-4, The jbosscmp-jdbc.xml Load Group Declaration

In *Listing 6-4*, two load groups are declared: basic and contact info. Note that the load groups do not need to be mutually exclusive. For example, both of the load groups contain the nickName field. In addition to the declared load groups, JBossCMP automatically adds a group named "*" (the star group) that contains every cmp-field and cmr-field with a foreign key in the entity.

Read-ahead

Optimized loading in JBossCMP is called read-ahead. This term was inherited from JAWS, and refer to the technique of reading the row for an entity being loaded, as well as the next several rows; hence the term read-ahead. JBossCMP implements two main strategies (on-find and on-load) to optimize the loading problem identified in the previous section.

The extra data loaded during read-ahead is not immediately associated with an entity object in memory, as entities are not materialized in JBoss until actually accessed. Instead, it is stored in the preload cache where it remains until it is loaded into an entity or the end of the transaction occurs. The following sections describe the read-ahead strategies.

on-find

The on-find strategy reads additional columns when the query is invoked. If the query in the scenario detailed in *Listing 6-1* is on-find optimized, JBossCMP will execute the following query at Arrow 1:

```
SELECT t0_g.id, t0_g.name, t0_g.nick_name, t0_g.badness
FROM gangster t0_g
ORDER BY t0_g.id ASC
```

Listing 6-5, on-find Optimized findAll Query

Then at Arrow 2, all of the required data would be in the preload cache, so no additional queries would be executed. This strategy is effective for queries that return a small amount of data, but becomes very inefficient when trying to load a large result set into memory.[18] The following table shows the execution of this query:

Table 6-2, on-find Optimized Query Execution

id	name	nick_name	badness	hangout	organization
0	Yojimbo	Bodyguard	7	0	Yakuza
1	Takeshi	Master	10	1	Yakuza
2	Yuriko	Four finger	4	2	Yakuza
3	Chow	Killer	9	3	Triads
4	Shogi	Lightning	8	4	Triads
5	Valentino	Pizza-Face	4	5	Mafia
6	Toni	Toothless	2	6	Mafia
7	Corleone	Godfather	6	7	Mafia

The read-ahead strategy and load-group for a query is defined in the query element. If a read-head strategy is not declared in the query element, the strategy declared in the entity element or defaults element is used. The on-find configuration follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findAll_onfind</method-name>
               <method-params/>
            </query-method>
            <jboss-ql><![CDATA[
               SELECT OBJECT(g)
               FROM gangster g
               ORDER BY g.gangsterId
            ]]></jboss-ql>
            <read-ahead>
               <strategy>on-find</strategy>
```

[18] JBossCMP uses soft references in the read-ahead cache implementation, so data will be cached and then immediately released.

```
                <page-size>4</page-size>
                <eager-load-group>basic</eager-load-group>
             </read-ahead>
          </query>
       </entity>
    </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 6-6, The jbosscmp-jdbc.xml on-find Declaration

One problem with the on-find strategy is that it must load additional data for every entity selected. Commonly in web applications only a fixed number of results are rendered on a page. Since the preloaded data is only valid for the length of the transaction, and a transaction is limited to a single web http hit, most of the preloaded data is not used. The on-load strategy discussed in the next section does not suffer from this problem.

on-load

The on-load strategy block loads additional data for several entities when an entity is loaded, starting with the requested entity and the next several entities in the order they were selected.[19] This strategy is based on the theory that the results of a find or ejbSelect will be accessed in forward order. When a query is executed, JBossCMP stores the order of the entities found in the list cache. Later, when one of the entities is loaded, JBossCMP uses this list to determine the block of entities to load. The number of lists stored in the cache is specified with the list-cache-max element of the entity. This strategy is also used when faulting in data not loaded in the on-find strategy. With this strategy, the query executed at Arrow 1 remains unchanged.

```
SELECT t0_g.id
FROM gangster t0_g
ORDER BY t0_g.id ASC
```

Listing 6-7, on-load (Unoptimized) findAll Query

If, for example, the page size is set to four, JBossCMP will execute the following two queries to load the name, nickName and badness fields for the entities:

```
SELECT id, name, nick_name, badness
FROM gangster
WHERE (id=0) OR (id=1) OR (id=2) OR (id=3)
```

19 This is the read-ahead technique from JAWS.

```
SELECT id, name, nick_name, badness
FROM gangster
WHERE (id=4) OR (id=5) OR (id=6) OR (id=7)
```

Listing 6-8, on-load Optimized Load Queries

The following table shows the execution of these queries:

Table 6-3, on-load Optimized Query Execution

id	name	nick_name	badness	hangout	organization
0	Yojimbo	Bodyguard	7	0	Yakuza
1	Takeshi	Master	10	1	Yakuza
2	Yuriko	Four finger	4	2	Yakuza
3	Chow	Killer	9	3	Triads
4	Shogi	Lightning	8	4	Triads
5	Valentino	Pizza-Face	4	5	Mafia
6	Toni	Toothless	2	6	Mafia
7	Corleone	Godfather	6	7	Mafia

As with the on-find strategy, on-load is declared in the read-ahead element. The on-load configuration follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findAll_onload</method-name>
               <method-params/>
            </query-method>
            <jboss-ql><![CDATA[
               SELECT OBJECT(g)
               FROM gangster g
               ORDER BY g.gangsterId
            ]]></jboss-ql>
            <read-ahead>
```

```
                <strategy>on-load</strategy>
                <page-size>4</page-size>
                <eager-load-group>basic</eager-load-group>
             </read-ahead>
          </query>
       </entity>
    </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 6-9, The jbosscmp-jdbc.xml on-load Declaration

none

The none strategy is really an anti-strategy. This strategy causes the system to fall back to the default lazy-load code, and specifically does not read-ahead any data or remember the order of the found entities. This results in the queries and performance shown at the beginning of this chapter. The none strategy is declared with a read-ahead element. If the read-ahead element contains a page-size element or eager-load-group, it is ignored. The none strategy is declared in the following configuration:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <query>
            <query-method>
               <method-name>findAll_none</method-name>
               <method-params/>
            </query-method>
            <jboss-ql><![CDATA[
               SELECT OBJECT(g)
               FROM gangster g
               ORDER BY g.gangsterId
            ]]></jboss-ql>
            <read-ahead>
               <strategy>none</strategy>
            </read-ahead>
         </query>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 6-10, The jbosscmp-jdbc.xml none Declaration

Loading Process

In the previous section several steps use the phrase "when the entity is loaded." This was intentionally left vague because the commit option specified for the entity and the current state of the transaction determine when an entity is loaded. The following section describes the commit options and the loading processes.

Commit Options

Central to the loading process are the commit options, which control when the data for an entity expires. JBoss supports four commit options A, B, C and D. The first three are described in section 10.5.9 of the **Enterprise JavaBeans Specification Version 2.0 Final Release** and the forth, D, is specific to JBoss. A detailed description of each commit option follows:

- **A**: JBossCMP assumes it is the sole user of the database; therefore, JBossCMP can cache the current value of an entity between transactions, which can result is substantial performance gains. As a result of this assumption, no data managed by JBossCMP can be changed outside of JBossCMP. For example, changing data in another program or with the use of direct JDBC (even within JBoss) will result in an inconsistent database state.

- **B**: JBossCMP assumes that there is more than one user of the database but keeps the context information about entities between transactions. This context information is used for optimizing loading of the entity.[20] This is the default commit option.

- **C**: JBossCMP discards all entity context information at the end of the transaction.

- **D**: This is a JBoss specific commit option. This option is similar to commit option A, except that the data only remains valid for a specified amount of time.

The commit option is declared in the jboss.xml file. For a detailed description of this file see the **JBoss 3.0 Quick Start Guide**. The following example changes the commit option to A for all entity beans in the application:

```
<jboss>
   <container-configurations>
      <container-configuration>
         <container-name>Standard CMP 2.x EntityBean</container-name>
         <commit-option>A</commit-option>
      </container-configuration>
   </container-configurations>
</jboss>
```

Listing 6-11, The jboss.xml Commit Option Declaration

Eager-loading Process

One of the most important changes in CMP 2.0 is the change from using class fields for cmp-fields to abstract accessor methods. In CMP 1.x, the container could not know which fields were required in a transaction, so the container had to eager-load every field when loading the

[20] In a future version, JBossCMP will be able to keep the current data of a commit option B entity between transactions and validate that the data is still current using last-update optimistic locking. For entities that contain a large amount of data, this will result in a significant performance enhancement.

bean. In CMP 2.x, the container creates the implementation for the abstract accessors, so the container can know when the data for a field is required. JBossCMP can be configured to eager-load only some of the fields when loading an entity, and later lazy-load the remaining fields as needed.

When an entity is loaded, JBossCMP must determine the fields that need to be loaded. By default, JBossCMP will use the eager-load-group of the last query that selected this entity. If the entity has not been selected in a query, or the last query used the none read-ahead strategy, JBossCMP uses the default eager-load-group declared for the entity. In the following configuration, the basic load group is set as the default eager-load-group for the GangsterEJB entity:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <load-groups>
            <load-group>
               <load-group-name>most</load-group-name>
               <field-name>name</field-name>
               <field-name>nickName</field-name>
               <field-name>badness</field-name>
               <field-name>hangout</field-name>
               <field-name>organization</field-name>
            </load-group>
         </load-groups>
         <eager-load-group>most</eager-load-group>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 6-12, The jbosscmp-jdbc.xml Eager Load Declaration

The eager loading process is initiated the first time a method is called on an entity in a transaction. A detailed description of the load process follows:

1. If the entity context is still valid, no loading is necessary, and therefore the loading process is done. The entity context will be valid when using commit option A, or when using commit option D, and the data has not timed out.

2. Any residual data in the entity context is flushed. This assures that old data does not bleed into the new load.

3. The primary key value is injected back into the primary key fields. The primary key object is actually independent of the fields and needs to be reloaded after the flush in step 2.

4. All data in the preload cache for this entity is loaded into the fields.

5. JBossCMP determines the additional fields that still need to be loaded. Normally the fields to load are determined by the eager-load group of the entity, but can be overridden if the entity was located using a query or cmr-field with an on-find or on-load read-ahead strategy. If all of the fields have already been loaded, the load process skips to step 7.

6. A query is executed to select the necessary column. If this entity is using the on-load strategy, a page of data is loaded as described in the on-load section. The data for the current entity is stored in the context and the data for the other entities is stored in the preload cache.

7. The ejbLoad method of the entity is called.

Lazy-loading Process

Lazy-loading is the other half of eager-loading. If a field is not eager-loaded, it must be lazy-loaded. When the bean accesses an unloaded field, JBossCMP loads the field and any field in a lazy-load-group of which the unloaded field is a member. JBossCMP performs a set join and then removes any field that is already loaded. An example follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>GangsterEJB</ejb-name>
         <load-groups>
            <load-group>
               <load-group-name>basic</load-group-name>
               <field-name>name</field-name>
               <field-name>nickName</field-name>
               <field-name>badness</field-name>
            </load-group>
            <load-group>
               <load-group-name>contact info</load-group-name>
               <field-name>nickName</field-name>
               <field-name>contactInfo</field-name>
               <field-name>hangout</field-name>
            </load-group>
         </load-groups>
         <lazy-load-groups>
            <load-group-name>basic</load-group-name>
            <load-group-name>contact info</load-group-name>
         </lazy-load-groups>
      </entity>
   </enterprise-beans>
</jbosscmp-jdbc>
```

Listing 6-13, The jbosscmp-jdbc.xml Lazy Load Group Declaration

When the bean provider calls getName() with this configuration, JBossCMP loads name, nickName and badness (assuming they are not already loaded). When the bean provider calls getNickName(), the name, nickName, badness, contactInfo, and hangout are loaded. A detailed description of the lazy-loading process follows:

1. All data in the preload cache for this entity is loaded into the fields.
2. If the field value was loaded by the preload cache the lazy-load process is finished.
3. JBossCMP finds all of the lazy load groups that contain this field, performs a set join on the groups, and removes any field that has already been loaded.
4. A query is executed to select the necessary column. As in the basic load process, JBossCMP may load a block of entities. The data for the current entity is stored in the context and the data for the other entities is stored in the preload cache.

Relationships

Relationships are a special case in lazy-loading because a cmr-field is both a field and query. As a field it can be on-load block loaded, meaning the value of the currently sought entity and the values of the cmr-field for the next several entities are loaded. As a query, the field values of the related entity can be preloaded on-find.

Again, the easiest way to investigate the loading is to look at a usage scenario. In this example, an html table is generated containing each gangster and their hangout. The example code follows:

```
public String createGangsterHangoutHtmlTable() throws FinderException {
   StringBuffer table = new StringBuffer();
   table.append("<table>");

   Collection gangsters = gangsterHome.findAll_onfind();
   for(Iterator iter = gangsters.iterator(); iter.hasNext(); ) {
      Gangster gangster = (Gangster)iter.next();
      Location hangout = gangster.getHangout();
      table.append("<tr>");
      table.append("<td>").append(gangster.getName()).append("</td>");
      table.append("<td>").append(gangster.getNickName()).append("</td>");
      table.append("<td>").append(gangster.getBadness()).append("</td>");
      table.append("<td>").append(hangout.getCity()).append("</td>");
      table.append("<td>").append(hangout.getState()).append("</td>");
      table.append("<td>").append(hangout.getZipCode()).append("</td>");
      table.append("</tr>");
   }

   table.append("</table>");
   return table.toString();
}
```

Listing 6-14, Relationship Lazy Loading Example Code

For this example, the configuration of the Gangster findAll_onfind query is unchanged from the on-find section. The configuration of the Location entity and Gangster-Hangout relationship follows:

```
<jbosscmp-jdbc>
   <enterprise-beans>
      <entity>
         <ejb-name>LocationEJB</ejb-name>
         <load-groups>
            <load-group>
               <load-group-name>quick info</load-group-name>
               <field-name>city</field-name>
               <field-name>state</field-name>
               <field-name>zipCode</field-name>
            </load-group>
         </load-groups>
         <eager-load-group/>
      </entity>
   </enterprise-beans>
   <relationships>
      <ejb-relation>
         <ejb-relation-name>Gangster-Hangout</ejb-relation-name>
         <foreign-key-mapping/>
         <ejb-relationship-role>
            <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>gangster-has-a-hangout</ejb-relationship-role-name>
            <key-fields/>
            <read-ahead>
               <strategy>on-find</strategy>
```

```
                <page-size>4</page-size>
                <eager-load-group>quick info</eager-load-group>
             </read-ahead>
          </ejb-relationship-role>
          <ejb-relationship-role>
             <!-- outdented to fit on a printed page -->
<ejb-relationship-role-name>hangout-for-a-gangster</ejb-relationship-role-name>
             <key-fields>
                <key-field>
                   <field-name>locationId</field-name>
                   <column-name>hangout</column-name>
                </key-field>
             </key-fields>
          </ejb-relationship-role>
       </ejb-relation>
    </relationships>
</jbosscmp-jdbc>
```

Listing 6-15, The jbosscmp-jdbc.xml Relationship Lazy Loading Configuration

At Arrow 1, JBossCMP will execute the following query:

```
SELECT t0_g.id, t0_g.name, t0_g.nick_name, t0_g.badness
FROM gangster t0_g
ORDER BY t0_g.id ASC
```

Listing 6-16, on-find Optimized findAll Query

Then at Arrow 2, JBossCMP executes the following two queries to load the city, state, and zip fields of the hideout:

```
SELECT gangster.id, gangster.hangout,
       location.city, location.st, location.zip
FROM gangster, location
WHERE (gangster.hangout=location.id) AND
       ((gangster.id=0) OR (gangster.id=1) OR
        (gangster.id=2) OR (gangster.id=3))
```

```
SELECT gangster.id, gangster.hangout,
       location.city, location.st, location.zip
FROM gangster, location
WHERE (gangster.hangout=location.id) AND
       ((gangster.id=4) OR (gangster.id=5) OR
        (gangster.id=6) OR (gangster.id=7))
```

Listing 6-17, on-find Optimized Relationship Load Queries

The following table shows the execution of the queries:

Table 6-4, on-find Optimized Relationship Query Execution

Gangster					Location			
id	name	nick_name	badness	hangout	id	city	st	zip
0	Yojimbo	Bodyguard	7	0	0	San Fran	CA	94108
1	Takeshi	Master	10	1	1	San Fran	CA	94133
2	Yuriko	Four finger	4	2	2	San Fran	CA	94133
3	Chow	Killer	9	3	3	San Fran	CA	94133
4	Shogi	Lightning	8	4	4	San Fran	CA	94133
5	Valentino	Pizza-Face	4	5	5	New York	NY	10017
6	Toni	Toothless	2	6	6	Chicago	IL	60661
7	Corleone	Godfather	6	7	7	Las Vegas	NV	89109

Transactions

All of the examples presented in this chapter have been defined to run in a transaction. Transaction granularity is a dominating factor in optimized loading because transactions define the lifetime of preloaded data. If the transaction completes, commits, or rolls back, the data in the preload cache is lost. This can result in a severe negative performance impact.

The performance impact of running without a transaction will be demonstrated with an example similar to *Listing 6-1*. This example uses an on-find optimized query that selects the first four gangsters (to keep the result set small), and it is executed without a wrapper transaction. The example code follows:

```
public String createGangsterHtmlTable_no_tx() throws FinderException {
   StringBuffer table = new StringBuffer();
   table.append("<table>");
   Collection gangsters = gangsterHome.findFour();
   for(Iterator iter = gangsters.iterator(); iter.hasNext(); ) {
      Gangster gangster = (Gangster)iter.next();
      table.append("<tr>");
      table.append("<td>").append(gangster.getName()).append("</td>");
      table.append("<td>").append(gangster.getNickName()).append("</td>");
      table.append("<td>").append(gangster.getBadness()).append("</td>");
      table.append("</tr>");
   }
   table.append("</table>");
   return table.toString();
}
```

Listing 6-18, No Transaction Loading Example Code

The query executed at Arrow 1 follows:

```
SELECT t0_g.id, t0_g.name, t0_g.nick_name, t0_g.badness
FROM gangster t0_g
WHERE t0_g.id < 4
ORDER BY t0_g.id ASC
```

Listing 6-19, No Transaction on-find Optimized findAll Query

Normally this would be the only query executed, but since this code is not running in a transaction, all of the preloaded data is thrown away as soon as findAll returns. Then at Arrow 2 JBossCMP executes the following four queries (one for each loop):[21]

```
SELECT id, name, nick_name, badness
FROM gangster
WHERE (id=0) OR (id=1) OR (id=2) OR (id=3)
```

```
SELECT id, name, nick_name, badness
FROM gangster
WHERE (id=1) OR (id=2) OR (id=3)
```

```
SELECT id, name, nick_name, badness
FROM gangster
WHERE (id=2) OR (id=3)
```

[21] It's actually worse than this. JBossCMP executes each of these queries three times; once for each cmp-field that is accessed. This is because the preloaded values are discarded between the cmp-field accessor calls.

```
SELECT name, nick_name, badness
FROM gangster
WHERE (id=3)
```

Listing 6-20, No Transaction on-load Optimized Load Queries

The following table shows the execution of the queries:

Table 6-5, No Transaction on-find Optimized Query Execution

This performance is much worse than read-ahead none because of the amount of data loaded from the database. The number of rows loaded is determined by the following equation:

This all happens because the transaction in the example is bounded by a single call on the entity. This brings up the important question "How do I run my code in a transaction?" The answer depends on where the code runs. If it runs in an EJB (session, entity, or message driven), the method must be marked with the Required or RequiresNew trans-attribute in the assembly-descriptor. If the code is not running in an EJB, a user transaction is necessary. The following code wraps a call to the method declared in *Listing 6-18* with a user transaction:

```
public String createGangsterHtmlTable_with_tx() throws FinderException {
   UserTransaction tx = null;
   try {
      InitialContext ctx = new InitialContext();
      tx = (UserTransaction) ctx.lookup("UserTransaction");
      tx.begin();

      String table = createGangsterHtmlTable_no_tx();

      if(tx.getStatus() == Status.STATUS_ACTIVE) {
         tx.commit();
      }
      return table;
   } catch(Exception e) {
      try {
         if(tx != null) tx.rollback();
      } catch(SystemException unused) {
         // eat the exception we are exceptioning out anyway
      }
      if(e instanceof FinderException) {
         throw (FinderException) e;
      }
      if(e instanceof RuntimeException) {
         throw (RuntimeException) e;
      }
      throw new EJBException(e);
   }
}
```

Listing 6-21, User Transaction Example Code

Appendix A

A. About The JBoss Group

The JBoss Group, LLC. is an Atlanta-based professional services company created by Marc Fleury, founder and lead developer of the JBoss J2EE-based open source web application server. The JBoss Group brings together core JBoss developers to provide services such as training, support and consulting, as well as management of the JBoss software and services affiliate programs. These commercial activities subsidize the development of the free core JBoss server. For additional information on The JBoss Group see the JBoss website at **http://www.jboss.org/jbossgroup/services.jsp**.

Appendix B

B. Defaults

JBossCMP global defaults are defined in the standardjbosscmp-jdbc.xml file of the server/<server-name>/conf/directory file in the JBoss 3.0 distribution. Each application can override the global defaults in the jbosscmp-jdbc.xml file. The default options are contained in a defaults element of the configuration file. An example of the defaults section follows:

```
<jbosscmp-jdbc>
   <defaults>
      <datasource>java:/DefaultDS</datasource>
      <datasource-mapping>Hypersonic SQL</datasource-mapping>
      <create-table>true</create-table>
      <remove-table>false</remove-table>
      <read-only>false</read-only>
      <read-time-out>300</read-time-out>
      <pk-constraint>true</pk-constraint>
      <fk-constraint>false</fk-constraint>
      <row-locking>false</row-locking>
      <preferred-relation-mapping>foreign-key</preferred-relation-mapping>
      <read-ahead>
         <strategy>on-load</strategy>
         <page-size>1000</page-size>
         <eager-load-group>*</eager-load-group>
      </read-ahead>
      <list-cache-max>1000</list-cache-max>
   </defaults>
</jbosscmp-jdbc>
```

Listing B-1, The jbosscmp-jdbc.xml Defaults Declaration

Each option can apply to entities, relationships, or both, and can be overridden in the specific entity or relationship. A detailed description of each option follows:

Table B-1, defaults Tags

Tag Name	Description	Required
datasource	This is the jndi-name used to look up the datasource. All database connections used by an entity or relation-table are obtained from the datasource.	No, default it java:/DefaultDS

Tag Name	Description	Required
datasource-mapping	This specifies the name of the type-mapping, which determines how Java types are mapped to SQL types, and how EJB-QL functions are mapped to database specific functions. Type-mapping is discussed in Appendix C.	No, default is Hypersonic SQL
create-table	If true, JBossCMP will attempt to create a table for the entity. When the application is deployed, JBossCMP checks if a table already exists before creating the table. If a table is found, it is logged, and the table is not created. This option is very useful during the early stages of development when the table structure changes often.	No, default is true
remove-table	If true, JBossCMP will attempt to drop the table for each entity and each relation-table mapped relationship when the application is undeployed. This option is very useful during the early stages of development when the table structure changes often.	No, default is false
read-only	If true, the bean provider will not be allowed to change the value of any fields. A field that is read-only will not be stored in, or inserted into, the database. If a primary key field is read-only, the create method will throw a CreateException. If a set accessor is called on a read-only field, it throws an EJBException. Read-only fields are useful for fields that are filled in by database triggers, such as last update. The read-only option can be overridden on a per cmp-field basis, which is discussed in Chapter 3.	No, default is false
read-time-out	This is the amount of time in milliseconds that a read on a read-only field is valid. A value of 0 means that the value is always reloaded at the start of a transaction, and a value of -1 means that the value never times out. This option can also be overridden on a per cmp-field basis. If read-only is false, this value is ignored.	No, default is 300
row-locking	If true, JBossCMP will lock all rows loaded in a transaction. Most databases implement this by using the SELECT FOR UPDATE syntax when loading the entity, but the actual syntax is determined by the row-locking-template in the datasource-mapping used by this entity.	No, default is false

Tag Name	Description	Required
preferred-relation-mapping	This is used to determine the default mapping for relationships. The valid options are foreign-key and relation-table. This option applies to relationships.	No, default is foreign-key
pk-constraint	If true, JBossCMP will add a primary key constraint when creating tables.	No, default is true
read-ahead	This controls caching of query results and cmr-fields for the entity. This option is discussed in Chapter 6.	No, see Chapter 6
list-cache-max	This specifies the number of read-lists that can be tracked by this entity. This option is discussed in Chapter 6.	No, default is 1000

Appendix C

C. Datasource Customization

JBossCMP includes predefined type-mappings for the following databases: Cloudscape, DB2, DB2/400, Hypersonic SQL, InformixDB, InterBase, MS SQLSERVER, MS SQLSERVER2000, mySQL, Oracle7, Oracle8, Oracle9i, PointBase, PostgreSQL, PostgreSQL 7.2, SapDB, SOLID, and Sybase. If you do not like the supplied mapping, or a mapping is not supplied for your database, you will have to define a new mapping. If you find an error in one of the supplied mappings, or if you create a new mapping for a new database, please consider posting a patch at the JBoss project page on **SourceForge**.

Type Mapping

A type-mapping is simply a set of mappings between Java class types and database types. The following is the current mapping of a short for Oracle 9i.

```
<jbosscmp-jdbc>
   <type-mapping>
      <name>Oracle9i</name>
      <mapping>
         <java-type>java.lang.Short</java-type>
         <jdbc-type>NUMERIC</jdbc-type>
         <sql-type>NUMBER(5)</sql-type>
      </mapping>
   </type-mapping>
</jbosscmp-jdbc>
```

Listing C-1, The jbosscmp-jdbc.xml Type Mapping Declaration

If JBossCMP cannot find a mapping for a type, it will serialize the object and use the java.lang.Object mapping. The following describes the three elements of the mapping element:

Table C-1, Type Mapping Tags

Tag Name	Description	Required
java-type	This is the fully qualified name of the Java class to be mapped. If the class is a primitive wrapper class such as java.lang.Short, the mapping also applies to the primitive type.	Yes

Tag Name	Description	Required
jdbc-type	This is the JDBC type that is used when setting parameters in a JDBC PreparedStatement or loading data from a JDBC ResultSet. The valid types are defined in java.sql.Types.	Yes
sql-type	This is the SQL type that is used in create table statements. Valid sql-types are only limited by your database vendor.	Yes

Function Mapping

EJB-QL and JBossQL contain eight functions: ABS, CONCAT, LENGTH, LCASE, LOCATE, SQRT, SUBSTRING, and UCASE. By default, these functions are mapped to JDBC SQL extension scalar functions. For example, CONCAT('Hot', 'Java') would map to {fn concat('Hot', 'Java')}. Several of the major database vendors do not support this function style in a pathetic effort to lock users into their database. The mapping for these functions can be overridden by adding function-mapping elements to the type-mapping element. The following is an example of the concat function mapping for Oracle 9i.

```xml
<jbosscmp-jdbc>
   <type-mapping>
      <name>Oracle9i</name>
      <function-mapping>
         <function-name>concat</function-name>
         <function-sql>(?1 || ?2)</function-sql>
      </function-mapping>
   </type-mapping>
</jbosscmp-jdbc>
```

Listing C-2, The jbosscmp-jdbc.xml Function Mapping Declaration

Appendix

D

D. Revision History

This appendix lists changes to this document and changes to JBossCMP.

Beta 1

Chapter 2 Entities

Changed type-mapping element in entity and defaults to datasource-mapping. This change was required to enable DTD validation.

Moved description of read-ahead to Chapter 6

Removed the debug element. Logging is now completely controlled by log4j.

Clarified interpretation of read-time-out element.

Changed select-for-update element to row-locking. Not all database vendors support the SELECT FOR UPDATE syntax, but most support some form of row locking.

Chapter 3 CMP-Fields

Moved description and specification of eager/lazy loading to Chapter 6.

Support for not-null columns has been added.

Chapter 4 Container Managed Relationships

Completely changed the mapping of relationships in ejb-jar.xml.

- Moved the ejb-relationship-role elements out of the foreign-key-mapping and relation-table-mapping elements.
- Merged foreign-key-fields and table-key-fields into a common key-fields element. Keys are now always defined in terms of the current entity. This changes nothing for relation-table-mapped relationships, but for foreign-key-mapped relationships, the key-field mapping is exactly backwards.

Added support for read-only relationships with the introduction of the read-only and read-time-out elements.

Added support for automatic foreign-key constraint generation with the addition of the fk-constraint element to ejb-relationship-role.

Added support for all table related configuration elements to relation-table-mapping.

Chapter 6 Optimized Loading

The read-ahead code has been completely rewritten.

The specification of eager/lazy loading groups has completely changed.

- Added named load groups which are referenced in the eager-load-group, lazy-load-groups, and read-ahead elements.
- The eager-load element has been replaced with the eager-load-group element, which only contains the name of the group to eager load by default.
- The lazy-load-groups element no longer contains lazy-load-group elements; rather it contains load-group-name elements.

A row-locking-template has been added to type-mapping elements to enable vendor specific row-locking syntax.

Beta 2

Chapter 5 Queries

Completely rewrote EJB-QL compiler using JavaCC.

- Removed restriction on mySQL usage of NOT EMPTY operator by adding support for LEFT JOIN instead of EXISTS subquery.
- Fixed the mapping of MEMBER OF and NOT EMPTY clause.

Added JBoss query language (JBossQL).

Added DynamicQL, which allows runtime compilation and execution of JBossQL.

Changed DeclaredSQL to allow finder to use DISTINCT.

Added the ability to specify the alias used for the main select table.