Oracle9*i*AS TopLink

CMP for Users of IBM WebSphere Server Guide

Release 2 (9.0.3)

August 2002

Part No. B10067-01

ORACLE

Oracle 9*i*AS TopLink CMP for Users of IBM WebSphere Server Guide, Release 2 (9.0.3)

Part No. B10067-01

Contents

3 Configuring TopLink Container-Managed Persistence

4 EJB Entity Bean Deployment

5 Defining and Executing Queries

6 Run time considerations

7 Customization

8 The Single Bean Example Application

A EJB Architectures Summary

Send Us Your Comments

Oracle 9*i*AS TopLink CMP for Users of IBM WebSphere Server Guide, Release 2 (9.0.3)

Part No. B10067-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: iasdocs_us@oracle.com
- FAX: 650-506-7407 Attn: Oracle9*i* Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle9*i* Application Server Documentation
  500 Oracle Parkway, M/S 2op3
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This section introduces the information you need to get the most out of the documentation that accompanies your software. This preface contains these topics:

Intended Audience

This document is intended for application developers who perform the following tasks:

- Application design and development
- Application testing and benchmarking
- Application integration

This document assumes that you are familiar with the concepts of object-oriented programming, the Enterprise JavaBeans (EJB) specification, and with your own particular Java development environment.

The document also assumes that you are familiar with your particular operating system (Windows, UNIX, or other). The general operation of any operating system is described in the user documentation for that system, and is not repeated in this manual.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Structure

This document contains:

Chapter 1, "Introduction"

This chapter provides an overview of the TopLink and CMP concepts that enable you to fully-leverage TopLink CMP.

Chapter 2, "Mapping Entity Beans"

This chapter describes how to map container-managed entity beans using the object mapping features of TopLink for Java. Instructions and hints for using direct and relationship mappings in an EJB context are provided, and differences between beans and regular Java objects are outlined.

Chapter 3, "Configuring TopLink Container-Managed Persistence"

This chapter describes the configuration and testing of TopLink Container-Managed Persistence.

Chapter 4, "EJB Entity Bean Deployment"

This chapter describes how to deploy beans within the application server.

Chapter 5, "Defining and Executing Queries"

This chapter describes the TopLink support for creating and customizing finders.

Chapter 6, "Run time considerations"

This chapter discusses some of the run-time issues associated with developing an application that uses TopLink Container-Managed Persistence.

Chapter 7, "Customization"

This chapter describes advanced customization of mappings, logins, and other aspects of persistence. These customizations enable you to take advantage of advanced TopLink features, JDBC driver features, or gain “low-level” access to some of TopLink for Java APIs that are normally masked.

Chapter 8, "The Single Bean Example Application"

This chapter introduces the basic concepts that are required to build and deploy an entity bean with TopLink.

Appendix A, "EJB Architectures Summary"

This appendix provides an overview of some of the basic design patterns available when using TopLink and TopLink CMP. It briefly suggests some of the more useful EJB designs and their suitability to specific applications.

Appendix B, "The toplink-ejb-jar DTD"

This appendix contains a listing of the `toplink-ejb-jar` document type description (DTD).

Related Documents

For more information, see these Oracle resources:

Oracle9*i*AS TopLink Getting Started

Provides installation procedures to install and configure TopLink. It also introduces the concepts with which you should be familiar to get the most out of TopLink.

Oracle9*i*AS TopLink Tutorials

Provides tutorials illustrating the use of TopLink. It is written for developers who are familiar with the object-oriented programming and Java development environments.

Oracle9*i*AS TopLink Foundation Library Guide

Introduces TopLink and the concepts and techniques required to build an effective TopLink application. It also gives a brief overview of relational databases and describes who TopLink accesses relational databases from the object-oriented Java domain.

Oracle9*i*AS TopLink Mapping Workbench Reference Guide

Includes the concepts required for using the TopLink Mapping Workbench, a stand-alone application that creates and manages your descriptors and mappings for a project. This document includes information on each Mapping Workbench function and option and is written for developers who are familiar with the object-oriented programming and Java development environments.

Oracle9*i*AS TopLink Container Managed Persistence for Application Servers

Provides information on TopLink container-managed persistence (CMP) support for application servers. Oracle provides an individual document for each application server specifically supported by TopLink CMP.

Oracle9*i*AS TopLink Troubleshooting

Contains general information about TopLink's error handling strategy, the types of errors that can occur, and Frequently Asked Questions (FAQs). It also discusses troubleshooting procedures and provides a list of the exceptions that can occur, the most probable cause of the error condition, and the recommended action.

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

Conventions

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.

Convention	Meaning	Example
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id` and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example
. . .	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS subquery;` `SELECT col1, col2, ... , coln FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose **Start** >	How to start a program.	To start the Oracle Database Configuration Assistant, choose **Start** > **Programs** > **...** .
Case sensitivity and file and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`
	IMPORTANT NOTE: File names and directory names *are* case sensitive under UNIX. Where the name of a file or directory is mentioned and the operating system is a non-Windows platform, you must enter the names exactly as they appear unless instructed otherwise.	

Convention	Meaning	Example
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`

	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"`
		`C:\>imp SYSTEM/`*`password`*
		`FROMUSER=scott TABLES=(emp, dept)`

<INSTALL_DIR>	Represents the Oracle home installation directory name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`SET CLASSPATH=`*`[INSTALL_DIR]`*`\jre\bin`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ▪ `C:\orant` for Windows NT ▪ `C:\orawin95` for Windows 95 ▪ `C:\orawin98` for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install Oracle9*i* release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora90`. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle9i Database Getting Starting for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory.

1

Introduction

This document includes an example application that illustrates how to use EJB 1.1 entity beans with TopLink Container-Managed Persistence. The Single Bean Example shows how a single EJB bean can be made persistent using TopLink Container-Managed Persistence's container-managed persistence support. This example application illustrates simple direct-to-field mappings and introduces the basic steps required to deploy a bean.

If you are a new user, go through the single bean example, as well as the other examples included in the TopLink installation. These examples provide you with some hands-on experience with TopLink, and give you a better understanding of TopLink's power and usefulness.

TopLink Container-Managed Persistence

TopLink Container-Managed Persistence is an extension of the TopLink for Java persistence framework. In addition to providing all of TopLink for Java's object-relational persistence facilities, TopLink Container-Managed Persistence also provides container-managed persistence (CMP) for Enterprise JavaBeans (EJBs) deployed in the IBM WebSphere server.

TopLink's CMP supports complex mappings from entity beans to relational database tables, and enables you to model relationships between beans, and between beans and regular Java objects. TopLink provides a rich set of querying options and allows query definition at the bean-level rather than the database level.

TopLink Container-Managed Persistence provides container-managed persistence and other object-relational mapping features for IBM WebSphere Server 4.0. Earlier versions of IBM WebSphere are not supported by this release.

TopLink Container-Managed Persistence supports the specification as defined by Sun Microsystems.

TopLink Container-Managed Persistence is an extension of the TopLink for Java product and shares all of its core functionality.

TopLink for Java

TopLink for Java provides an easy way to map a Java object model to a relational database. TopLink is a persistence framework that bridges the gap between objects and relational databases, and allows you to work at the object level.

TopLink supports the creation of a wide variety of Java applications. For building two-tier, three-tier, or n-tier applications; TopLink can be used within EJB and non-EJB environments. It can also be used within Java application servers or on its own.

If you are using TopLink for persistence requirements other than container-managed persistence (such as traditional two-, three-, or *n*-tier applications, non-EJB applications, or session bean-based applications), refer to the *Oracle9iAS TopLink Foundation Library Guide*. That document includes information on advanced TopLink features that are not included in this manual.

TopLink Mapping Workbench

TopLink Mapping Workbench is a separate tool that provides a graphical method of configuring the descriptors and mappings of a project. It provides many checks to ensure that the descriptor settings are valid, and it also provides advanced functionality for accessing the database and creating a database schema.

The TopLink Mapping Workbench does not generate Java code during development, which would be unmanageable if the descriptors changed. Instead, it stores descriptor information in an XML deployment file, which can be read into a Java application using a TopLink method. When the application needs to be repackaged into a runtime, TopLink can then generate a `.java` file from the XML file, eliminating the need for TopLink Mapping Workbench files at runtime.

The TopLink Mapping Workbench displays all of the project information for a given project, including classes and tables. Refer to the *Oracle9iAS TopLink Mapping Workbench Reference Guide* for more information on editing projects and descriptors using TopLink Mapping Workbench.

Understanding container-managed persistence

This section introduces the concepts required to use TopLink's container-managed persistence (CMP) facilities. It highlights the particular features available in TopLink Container-Managed Persistence that are not available in TopLink's core Java Foundation Library and explains any differences in the use of other core features.

Enterprise JavaBeans (EJBs)

This manual assumes that you have some familiarity with Enterprise JavaBeans (EJBs) and related concepts. This section provides an overview of some of the key terms that are encountered when discussing EJBs.

For more information about Enterprise JavaBeans, visit the Sun Microsystems EJB site at `http://java.sun.com/products/ejb`.

Terminology and definitions

Enterprise JavaBeans To quote the Sun EJB specification, an enterprise bean implements a business task, or a business entity. Enterprise JavaBeans are server-side domain objects that fit into a standard component-based architecture for building enterprise applications using the Java language. They are Java objects that, when installed in an EJB server such as the IBM WebSphere Server, become distributed, transactional, and secure components. There are two kinds of EJBs: session beans and entity beans.

EJB Server and Container An EJB bean is said to reside within an EJB Container that in turn resides within an EJB Server. The exact distinction between container and server is not completely defined. In general, the server provides the bean with access to various services (transactions, security, and so on.) while the container provides the execution context for the bean by managing its life cycle.

Deployment descriptors The additional information required to install an EJB within its server is provided in the *deployment descriptors* for that bean. The deployment descriptors consists of a set of XML files that provide all of the required security, transaction, relationship, and persistence information for the bean.

Session beans Session beans represent a business operation, task, or process. Although the use of a session bean may involve database access, the beans are not in themselves persistent – they do not directly represent a database entry. Session beans may or may not retain conversational state; they may be stateful and retain

client information between calls, or they may be stateless and only retain information within a single method call.

TopLink may be used with session beans to make the regular Java objects that they access persistent, or can be used to access TopLink persistent entity beans. Session beans may also act as wrappers to other legacy applications.

Entity beans Entity beans represent a persistent data object – an object with durable state that exists from one access to the next. To accomplish this, the entity bean must be made persistent in a relational database, object database, or some other storage facility.

Two schemes exist for making entity beans persistent: bean-managed persistence (BMP) and container-managed persistence (CMP). BMP requires that the bean developer hand-code the methods that perform the persistence work. CMP uses information supplied by the developer or deployer to handle all aspects of persistence.

Java objects and Entity Beans

A Java object contains the following components:

Attributes. Store primitive data such as integers, and also store simple Java types such as String and Date.

Relationships References to other TopLink-enabled classes. A TopLink-enabled class has a descriptor and can be stored in the database. Because TopLink-enabled classes can be stored in a database, they are called persistent classes.

Methods Paths of execution that can be invoked in a Java environment. Methods are not stored in the database because they are static.

An entity bean has the following parts:

The bean instance An instance of an entity bean class supplied by the developer of the bean. It is a regular Java object whose class implements the `javax.ejb.EntityBean` interface. The bean instance has persistent state. The client application should never access the bean instance directly.

The EJBObject An instance of a generated class that implements the remote interface defined by the bean developer. This instance wraps the bean and all client interaction is made through this object. The EJBObject does not have persistent state.

The EJBHome An instance of a class that implements the home interface supplied by the bean developer. This instance is accessible from JNDI and provides all create and finder methods for the EJB. The EJBHome does not have persistent state.

The EJB Primary Key An instance of the primary key class provided by the bean developer. The primary key is a serializable object whose fields match the primary key fields in the bean instance. Although the EJB Primary Key shares some data with the bean instance, it does not have persistent state. Note that as of EJB 1.1, it is not required that a bean have a separate primary key class when the key consists of a single field.

For more information about IBM WebSphere Server tools, APIs, or concepts, refer to the IBM WebSphere Server documentation.

For more information about the Enterprise JavaBeans standard, visit the Sun Microsystems EJB site at `http://java.sun.com/products/ejb`.

2

Mapping Entity Beans

This chapter describes how to map container-managed entity beans using the object mapping features of TopLink for Java. Instructions and hints for using direct and relationship mappings in an EJB context are provided, and differences between beans and regular Java objects are outlined.

For information on direct and relationship mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*. You should read and thoroughly understand those chapters before attempting to map entity beans.

Using TopLink Mapping Workbench

When using TopLink Mapping Workbench with entity beans, the bean classes themselves should be loaded into TopLink Mapping Workbench. The remote, local, home, and local home interfaces and the primary key class do not need to be loaded, nor should mappings be defined using these classes.

Make sure you include any classes referred to by the entity beans on the project classpath that is used by the TopLink Mapping Workbench project, otherwise errors may occur when the beans are loaded. The remote, local, home, and localhome interfaces should also be avilable on the classpath, as they may be used during EJB validation.

Mappings

TopLink mappings define how an object's attributes are to be represented in the database. Attributes that are to be persistent, or that reference other beans or mapped objects, must be mapped to the database using either direct or relationship mappings.

To enable container-managed persistent storage of entity beans, the attributes on the bean implementation class must be mapped. The home and remote interface classes should not be mapped. Primary key classes, if they exist, also should not be mapped.

Creating mappings

You can create mappings by using TopLink Mapping Workbench or by using the Java code-based API. TopLink Mapping Workbench is a visual tool that offers windows and dialogs to set properties and to configure the mappings and TopLink descriptors for any given project. This is the preferred method of creating mappings, and should be used whenever possible.

TopLink Mapping Workbench imposes some limitations that require you to use the code API instead of the tool, but these limitations are few and are mentioned in the TopLink Mapping Workbench documentation.

For more information on the TopLink Mapping Workbench features and usage, and on the limitations mentioned above, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

Direct mappings

Direct mappings define how a persistent object refers to objects that do not have TopLink descriptors, such as the JDK classes, primitive types and other non-persistent classes.

Attributes containing state that is a primitive object, or a regular object that is not itself mapped to the database should be mapped using a direct mapping. For example, a String attribute would need a direct to field mapping for the attribute to be stored in a VARCHAR field.

For a complete description of direct mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

Entity bean attributes can be mapped using direct mappings without any special considerations.

Note: The entity context attribute (type `javax.ejb.EntityContext`) should not be mapped.

Relationship mappings

Persistent objects use *relationship mappings* to store references to instances of other persistent classes. The appropriate mapping type is based primarily upon the cardinality of the relationship (for example, one-to-one compared to one-to-many). For a complete description of relationship mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

Entity beans may be related to regular Java objects, other entity beans, or both. The following sections outline the mappings and conditions where special attention must be paid to correctly map beans and execute operations that traverse or modify these relationships.

Mappings between entity beans

The EJB 1.1 specification does not specify how one entity bean should store an object reference to another entity bean. TopLink for IBM WebSphere goes beyond what is available in the specification and allows the creation of inter-bean relationships.

A bean that has a relationship to another bean acts as a "client" of that bean; that is, it does not access the actual bean directly but acts through the remote interface of the bean. For example, if an `OrderBean` is related to a `CustomerBean`, it has an instance variable of type `Customer` (the remote interface of the `CustomerBean`) and only accesses those methods defined on the Customer interface.

Note: Although beans must refer to each other through their `remote` interface, all TopLink descriptors and projects refer to the bean class. For example, if you are mapping beans using the TopLink Mapping Workbench and defining relationships between them, you need to load only the bean classes and not the `remote` or `home` interfaces. When defining a relationship mapping in both the TopLink Mapping Workbench and code API, the "reference class" is always the bean class.

Mappings between entity beans and Java objects

The EJB 1.1 specification notes that entity beans should represent "independent business objects" and that *dependent objects* are "better implemented as a Java class (or several classes) and included as part of the entity bean on which it depends."

The following relationship mappings may exist between an entity bean and regular Java objects:

- One-to-one, privately-owned mappings (bean is source, Java object is target)
- One-to-many, privately-owned mappings (bean is source, Java object(s) is target)
- Aggregate mappings (bean is source, Java object is target)
- Direct collection mappings (bean is source, Java object is target and is a "base" datatype, such as String, or Date)

Relationships from entity beans to regular Java objects should be dependent and relationships between entity beans should be independent.

If dependent objects are exposed to the client, these objects must be serializable.

One-to-one mappings

One-to-one mappings represent simple pointer references between two objects. For a complete description of one-to-one mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

One-to-one mappings are valid between entity beans, or between an entity bean and a regular Java object where the entity bean is the source and the regular Java object is the target of the relationship.

To maintain EJB compliance, the object attribute that points to the target of the relationship must be of the correct type if the target is a bean. This must be the remote interface type and not the bean class.

There are a number of advanced variations on one-to-one mappings, that allow for more complex relationships to be defined — in particular *variable one-to-one mappings* allow for polymorphic target objects to be specified. These variations are not available for entity beans, but are valid for dependent Java objects. For more information on these kinds of mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

One-to-many mappings

One-to-many mappings are used to represent the relationship between a single source object and a collection of target objects. For more information on one-to-many mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

One-to-many mappings are valid between entity beans or between an entity bean and a collection of privately-owned regular Java objects.

As described in the *Oracle9iAS TopLink Mapping Workbench Reference Guide*, a one-to-one mapping should also be created from the target object back to the source. The object attribute that contains a pointer to the bean must be of the correct type (the local interface type) and not the bean class.

Many-to-many mappings

Many-to-many mappings represent the relationships between a collection of source objects and a collection of target objects. They require the creation of an intermediate table for managing the associations between the source and target records. For more information on many-to-many mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

When using container-managed persistence, many-to-many mappings are valid only between entity beans and cannot be privately owned. The exception is when a many-to-many mapping is used to implement a logical one-to-many mapping with a relation table.

Aggregate object mappings

Two objects are related by aggregation if there is a strict one-to-one relationship between the objects and all the attributes of the second object can be retrieved from the same table(s) as the owning object. This means that if the target (child) object exists, then the source (parent) object must also exist. The child (owned object) cannot exist without its parent.

For a complete description of aggregate object mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide.*

Aggregate mappings can be used with entity beans when the source of the mapping is an entity bean and the target is a regular Java object. It is not valid to make an entity bean the target of an aggregate object mapping. As a consequence, it follows that aggregate mappings between entity beans are likewise invalid.

Note: Aggregate objects are privately owned and should not be shared or referenced by other objects.

Aggregate collection mappings

Aggregate collection mappings are used to represent aggregate relationships between a single source object and collection of target objects. Unlike normal one-to many mappings, there is no one-to-one back reference required. Unlike the normal aggregate mappings, a target table is required for the target objects.

For a complete description of Aggregate collection mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide.*

Aggregate collection mappings can be used with entity beans if the source of the relationship is an entity or Java object, and the targets of the mapping are regular Java objects. It is not possible to define an aggregate collection mapping with entity beans as the targets.

Aggregate collections are most appropriate when the target collections are expected to be moderate in size and a one-to-one mapping from target to source would be difficult. In addition, great care should be taken to ensure the identity of the Aggregate object, when referencing objects from an Aggregate within an Aggregate Collection.

Caution: Although aggregate collection mappings appear similar to one-to-many mappings, aggregate collections should not be used in place of one-to-many mappings. One-to-many mappings are more robust and scalable, and offer better performance. In addition, aggregate collections are privately owned by the source of the relationship and should not be shared or referenced by other objects.

Sequencing with Entity Beans

Sequencing is a mechanism which can be used to populate the primary key attribute of new objects/entity beans before inserting them into the database. Refer to the *Oracle9iAS TopLink Mapping Workbench Reference Guide* for details on the different kinds of TopLink sequencing: table and native.

The configuration of sequencing is similar for both Java objects and entity beans. However, with entity beans a `create()` method exists on the bean home interface, and `ejbCreate()` and `ejbPostCreate()` methods are implemented on the bean implementation class.

Because the primary key is automatically generated, no primary key is passed into the `create()` method on the home interface when the bean is created. If you are using table-based sequencing or native sequencing for databases that support

pre-allocation of sequence numbers, the bean's primary key is available in the `ejbPostCreate()` method (which is the only native sequencing available for some CMP implementations).

Adding sequencing outside of the TopLink Mapping Workbench

WebSphere does not support auto-incrementing identity fields such as those found in DB2, Sybase and SQL Server databases when using TopLink for WebSphere Foundation Library. Use a sequence table or Oracle sequence object to implement sequencing with the TopLink for WebSphere Foundation Library.

In order for TopLink to correctly assign the sequence number to a newly created bean the following line must be added to the ejbCreate method in the bean class:

```
oracle.toplink.ejb.cmp.was.SessionLookupHelper.
getHelper().getSession(this).getActiveUnitofWork().assignSequenceNumber(this);
```

This line looks up the correct session and uses it to assign a sequence number to the bean. For more information on setting up sequencing see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

> **Note:** TopLink defers database access to the commit stage of the transaction. Clients that use client-initiated `UserTransactions` should not reference the primary key of newly created objects because they are not yet properly initialized before commit-time.

Inheritance

Although inheritance is a standard tool in object-oriented modeling, no implementation guidelines are outlined in the EJB specification. The EJB 1.0 specification does not address the issue, and the 1.1 specification discusses it only in general terms. As a result, any use of inheritance should be approached cautiously.

Some restrictions apply to entity beans when using inheritance:

- The home interfaces cannot inherit. The `findByPrimaryKey` method must be overloaded in order to have the correct return type, but this is not allowed. As a result, inheritance is not applicable to the home interfaces.
- The primary key of the subclass must be the same as that of the parent class.

The advanced example application illustrates inheritance. For more information, see the ReadMe.html file in the root directory of the advanced example application. This application is located in `{ORACLE_HOME}\TopLink\examples\was\examples\ejb\cmp11\advanced`.

Indirection

TopLink provides several mechanisms for just-in-time reading of relationships (also referred to as “lazy-loading” and “indirection”). There are two techniques that are available:

- use of indirection objects
- transparent indirection

While these indirection mechanisms are described in the *Oracle9iAS TopLink Mapping Workbench Reference Guide*, there are a number of issues that entity bean developers should be aware of when using indirection. In general these issues arise due to the migration of objects between client and server.

Issues include:

- Un-instantiated ValueHolders (indirection objects) do not survive serialization. If a ValueHolder is sent from the server to the client, it will no longer function unless it has been previously triggered.
- ValueHolders can be used in bean-bean relationships, and bean-object relationships, but should be avoided in relationships whose source is likely to be serialized to the client.
- Collections that use transparent indirection should not be serialized to the client application before they are instantiated. These collections will not function if they are serialized.
- ValueHolders should generally be used for bean-bean relationships, and for bean-object relationships. Transparent indirection can be used for collections that are not exposed to the client application.

For more information about these and other important issues, consult Chapter 6, "Run time considerations".

3

Configuring TopLink Container-Managed Persistence

This chapter describes the configuration and testing of TopLink Container-Managed Persistence. Please refer to *Oracle9iAS TopLink Getting Started* for installation information.

Software requirements

TopLink Container-Managed Persistence requires:

- IBM WebSphere 4.0
- A JDBC driver that is configured to connect with your local database system (see your database administrator)
- A Java development environment that is compatible with the JDBC API, such as:
  - Oracle JDeveloper
  - IBM WebSphere Studio Application Developer (WASD)
  - Sun JDK 1.2 or higher (note that TopLink ships with JDK 1.3 JAR files that are compatible with the JDK 1.2)
  - IBM VisualAge for Java
  - Any other Java environment that is compatible with the Sun JDK 1.3 or higher
- A command-line Java virtual machine (VM) executable (such as `java.exe` or `jre.exe`)

A note about the WebSphere Module Visibility setting TopLink only supports the WebSphere APPLICATION mode and MODULE mode for module visibility. This is because of the way WebSphere Application Server defines its class loader isolation mode for each setting.

A J2EE application (EAR file) can have multiple EJB modules (EJB JAR files). TopLink Container-Managed Persistence is designed to load one `toplink-ejb-jar.xml` per EJB module (EJB JAR). If the module visibility mode is not set to MODULE, an EJB module could load the wrong `toplink-ejb-jar.xml` from the other EJB module. TopLink CMP also supports the application with the restriction that each application can only have one EJB JAR file. Again, this is due to how class loader is designed for this mode. For more information about module visibility in WebSphere, consult the WebSphere documentation.

WebSphere 4.0 document from the link above indicates, "Portable J2EE applications should be written with Module-level visibility." Developer should keep this in mind when developing application.

Table 3–1 TopLink support of server-installable applications on server vs. module visibility mode

	Module Visibility Mode			
Installable Applications on Server	**Application**	**Module**	**Compatibility**	**Server**
Multiple applications in which each application can have multiple TopLink EJB modules	No	Yes	No	No
Multiple applications in which each application has single TopLink EJB module	Yes	Yes	No	No
Single application has multiple TopLink EJB modules	No	Yes	No	No
Single application has Single TopLink EJB module	Yes	Yes	Yes	Yes

Configuring TopLink CMP

The following procedures configure TopLink Container-Managed Persistence Foundation Library. This procedure assumes you have already installed TopLink Container-Managed Persistence.

Notes:

- If you are running under Windows NT, make sure you have administrator privileges. Also, make sure you modify the System Variables, not the User Variables.
- Java package names are case-sensitive. If you are installing under a 32-bit Windows environment, ensure the case sensitivity is enabled.

The steps required for preparing a system for TopLink depends on the type of system you are running:

- To prepare a Windows-based system, you must copy some JARs from the TopLink installation to your WebSphere installation (see ""Installing TopLink JARs to a WebSphere Server" on page 3-3, next) and then modify your PATH and CLASSPATH (see "To Modify the PATH and CLASSPATH for running the Deploy tool" on page 3-4).
- To prepare a non-Windows environment, you must modify your PATH and CLASSPATH.

Installing TopLink JARs to a WebSphere Server

1. Create a directory structure in the WebSphere installation directory as follows:

   `<WEBSPHERE_INSTALL_DIR>\lib\app`

   If you are using WebSphere, (the default installation directory is `C:\WebSphere\AppServer`) ; if you are using WSAD, the default directory is `C:\Program Files\ibm\Application Developer\plugins\com.ibm.etools.websphere.runtime`.

2. Copy the following JAR files to the directory you created:
   - *<INSTALL_DIR>*`\core\lib\toplink.jar`
   - *<INSTALL_DIR>*`\core\lib\xerces.jar`
   - *<INSTALL_DIR>*`\core\lib\tl_wasx.jar`

- *<INSTALL_DIR>*\hsql\lib\hsql_ds.jar
- *<INSTALL_DIR>*\hsql\lib\hsqldb.jar

where *<INSTALL_DIR>* is the directory into which you installed TopLink (C:\{ORACLE_HOME}\toplink if you installed to the default directory).

3. Check the PATH and, if necessary, modify it to include <JAVADIR>\bin where <JAVADIR> is the installation drive and directory for your Java Virtual Machine (VM) executable. This path must be the first one listed in the PATH environment variable, before any other paths.
4. Check and if necessary, modify the CLASSPATH environment variable.

> **Note:** The CLASS and CLASSPATH commands must appear in your autoexec.bat file. If either of them is missing, you must add them manually.

To Modify the PATH and CLASSPATH for running the Deploy tool

1. If you have not already done so, edit your PATH to include <JAVADIR>\bin\; as the first entry in the PATH list.
2. Edit the CLASSPATH to include all of the following:

   <INSTALL_DIR>\core\lib\toplink.jar;*<INSTALL_DIR>*\core\lib\xerces.jar;
   <INSTALL_DIR>\was_cmp\lib\tl_wasx.jar;*<INSTALL_DIR>*\hsql\lib\hsqlds.jar;
   <INSTALL_DIR>\hsql\lib\hsqldb.jar
   where *<INSTALL_DIR>* is the directory into which you installed TopLink (C:\{ORACLE_HOME}\toplink if you installed to the default directory).

Testing your TopLink installation

Compile and execute the JDBCConnectTest class in the oracle.toplink.tutorials.gettingstarted package.

To test the installation, do one of the following:

- Run the JDBCConnectTest class from the command line, passing appropriate database login information as parameters as follows:

  ```
  java oracle.toplink.tutorials.gettingstarted.JDBCConnectTest <username>
  <password> <database url> <jdbc driver class>
  ```

  or

- Modify the `main()` method to contain appropriate database login information as parameters, then recompile and execute the class. For example:

```
public static void main(String[] args) {
    JDBCConnectTest test = new
    JDBCConnectTest();
    if (args.length > 0) {
    test.connect(args[0], args[1], args[2], args[3]);
    }
    // Add your login information below
    else {
    /*This test uses the WebSphere sql server driver. Any compliant jdbc
    driver can be used. Do not use WebSphere pool devices for this test.*/
    test.connect("<user>","<password>",
    "WebSphere.jdbc.mssqlserver4.Driver",
    "jdbc:WebSphere:mssqlserver4:myserver:1433");
    }
}
```

If the code does not run successfully, check the error message and ensure that all of your settings are correct. You may also need to consult *Oracle9iAS TopLink Troubleshooting*.

Testing TopLink deployment tool

With the Java VM PATH and CLASSPATH properly configured, run the following classes to test the deployment tool:

```
java oracle.toplink.ejb.cmp.was.deploy.Deploy
```

```
java oracle.toplink.ejb.cmp.was.deploy.EJBDeployFrame(GUI
tool)
```

Other ways to test the installation:

- Run the deploy tool by clicking **Start** > **Programs** > **Oracle9iAS TopLink** > **Tools** > **Deploy Tool for WebSphere Server**.
- Execute the script deployTool.cmd or deployTool.sh to invoke the GUI tool:
  - *<INSTALL_DIR>*`\was_cmp\deployTool.cmd` (for Windows)
  - *<INSTALL_DIR>*`/was_cmp/deployTool.sh` (for Unix)

Testing TopLink Container-Managed Persistence with entity beans

To test TopLink Container-Managed Persistence with entity beans, run the Single Bean example documented in Chapter 8, "The Single Bean Example Application". .

When the `TopLinkConnectTest`, the TopLink Mapping Workbench, and the Single Bean example all run successfully, your TopLink installation is complete.

Running the Server with TopLink

Start the server as described in the WebSphere documentation. Refer to the WebSphere documentation for more information on class loader and classpath issues. Once the server is running, start the TopLink CMP application.

Note: If you encounter problems running WebSphere, contact IBM support.

See *Oracle9iAS TopLink Troubleshooting* for more information.

4

EJB Entity Bean Deployment

TopLink Container-Managed Persistence provides container-managed persistence (CMP) for 1.1 Enterprise JavaBeans (EJBs). The deployment process generates CMP code that allows TopLink to handle persistence aspects of EJBs. To install entity beans within the IBM WebSphere ApplicationServer and make them available for client applications, entity beans must be *deployed* within the server.

Overview of deployment

The goal of deployment is to make entity beans available to client applications. The early stages of the process involves writing entity beans, and mapping the beans to create a TopLink project. The deployment process involves several stages that start with configuring the deployment descriptor and generating deployed code in TopLink. The final stages of deployment process include deploying beans to the server and starting the beans. For much of the deployment process, WebSphere tools for Java, and TopLink Deployment Tool programs are used.

Understanding Deployment

The term "deployment" can sometimes cause confusion since there are actually a number of stages that occur between creating the bean classes and installing them in a running server.

Generally speaking the deployment process is three distinct steps:

1. **Configuration** - A number of properties are specified for the bean, including what persistence mechanism is being used and additional information required by the persistence mechanism.

2. **Code generation -** The information provided in the configuration stage is used by both IBM WebSphere and TopLink tools to generate the classes required for the bean. This includes helper classes related to transactions, persistence, and security, the `EJBHome` and `EJBObject` implementations, and the stubs and skeletons required for RMI-IIOP.
3. **Installation -** The server is started and instructed to make the bean available to clients.

Requirements before deployment

The following tasks must be completed prior to the deployment of TopLink persisted entity beans:

- Write and compile the various parts of each entity bean to be deployed, including the bean class, remote interfaces, home interfaces, and the primary key class (if required).
- Map the entity beans to the appropriate database tables, and save the mapping information in a TopLink project class or project file (deployable XML file).

Assemble the entity beans into a .jar or .ear file

To create the EAR or JAR file

1. Create a `toplink-ejb-jar.xml` file to associate the TopLink project with the JAR and put it in the same directory as `ejb-jar.xml`.
2. Add the Toplink project. If the project is a Java class then it must be added to the same directory location as bean classes. If the project it is an XML file then it must be added to the same directory as `ejb-jar.xml` file (i.e `META-INF/`).

Configuring entity bean deployment descriptors

The deployment descriptor and other WebSphere configuration settings can be edited by opening the EJB Jar or EAR file in the WebSphere Application Assembly Tool (WAAT; see the documentation accompanying WAAT for more information). The deployment descriptor can also be edited manually by opening it in a text editor.

```
<?xml version = "1.0" encoding = "US-ASCII"?>
<!DOCTYPE toplink-ejb-jar PUBLIC "-//Oracle Corp.
//DTD TopLink 9.0.3 CMP for WebSphere//EN"
"toplink-was-ejb-jar_903.dtd">
```

```
<toplink-configuration>
    <session>
        <name>ejb_cmp11_singlebean</name>
        <project-class>
            examples.ejb.cmp11.singlebean.AccountProject
        </project-class>
        <session-type>
            <server-session>
        </session-type>
        <login>
            <platform-class>
               oracle.toplink.internal.databaseaccess.HSQLPlatform
            </platform-class>
            <uses-external-connection-pool>true</uses-external-connection-pool>
            <uses-external-transaction-controller>true
            </uses-external-transaction-controller>
        _</login>
        <external-transaction-controller-class>
            oracle.toplink.jts.was.JTSExternalTransactionController_4_0
        </external-transaction-controller-class>
        <enable-logging>true</enable-logging>
        <logging-options>
            <log-debug>true</log-debug>
            <log-exceptions>true</log-exceptions>
            <log-exception-stacktrace>true</log-exception-stacktrace>
            <print-thread>true</print-thread>
            <print-session>true</print-session>
            <print-connection>true</print-connection>
            <print-date>true</print-date>
        </logging-options>
    </session>
</toplink-configuration>
```

Preparing for deployment

After the bean has been mapped to the appropriate tables using the TopLink Mapping Workbench, some additional configuration is required. This includes creating and editing the deployment descriptor information, and generating the classes that the WebSphere server will use at runtime.

This section describes the steps involved in preparing for deployment. Consult WebSphere Application Server documentation for the most up-to-date information on the reference tools.

The steps are:

- Run the TopLink Deployment tool to generate a deployed JAR composed of the EJB JAR files
- Deploy the deployed JAR to the server (stand-alone module), or assemble the deployed EJB JAR into an EAR file with WAAT and deploy the EAR file to the server (application mode).
- Start the application to make the beans available to the clients.

Running the Deployment Tool

The Deployment Tool is used to create the Deployed JAR from the EJB JAR or the VisualAge Deployed JAR. There are two ways to run the deployment tool: as a command line tool, or as a graphic interface.

Running the command line Deployment Tool

The command line deploy tool requires that the CLASSPATH is setup correctly. The Installation section in *Oracle9iAS TopLink Getting Started* contains a list of CLASSPATH items that the Deploy Tool requires. In addition, the dependent classes referred to by your entity beans must be in the CLASSPATH. The command-line deploy tool can be invoked using the following command:

```
java oracle.toplink.ejb.cmp.was.deploy.Deploy
```

Running the graphic Deployment tool

To run the graphic user interface deployment tool from Windows, click **Start** > **Programs** > **Oracle9iAS TopLink** > **Tools** > **Deploy Tool for WebSphere Server**.

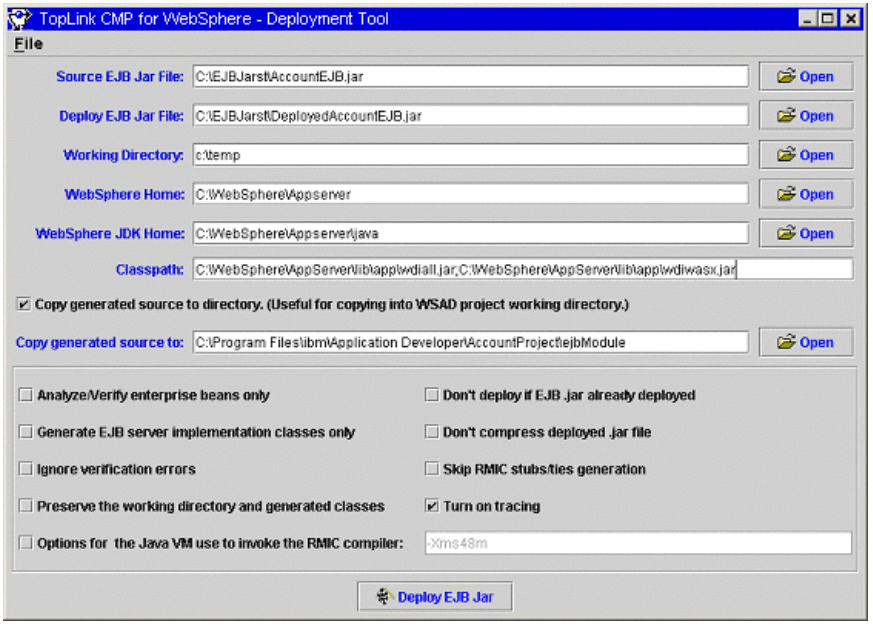
You can also manually start the script files that preconfigure the CLASSPATH to start this graphic interface deploy tool. The files are

`<INSTALL_DIR>\was_cmp\deployTool.cmd` (for Windows)

`<INSTALL_DIR>/was_cmp/deployTool.sh` (for Unix)

The Deployment Tool is used to create the Deployed JAR from the EJB JAR. Fill in the necessary fields in the Deploy Tool. The File menu allows settings of fields to be saved to a file and the settings are retrieved when the file is loaded.

Figure 4–1 The TopLink Deploy Tool



Set the Source EJB JAR File The EJB JAR created in WebSphere AAT or WSAD.

Deploy EJB JAR File The path and name of the JAR file which will be generated by the tool.

Working Directory The temporary directory in which generating classes will be stored. The directory will not be deleted if the “Preserve the working directory and generated classes” option is selected.

WebSphere Home The directory where WebSphere Application Server is installed, e.g. `C:\WebSphere\AppServer`.

WebSphere JDK Home The directory where the Java VM of WebSphere Application Server is installed, e.g. `C:\WebSphere\AppServer\java`

Classpath Set the Classpath field to include the `toplink.jar` and the `tl_wasx.jar` files and other required resources such as the dependent Java classes.

To deploy a JAR

1. If the **Copy generated source to directory** option is selected, a copy of the generated code is placed in the specified directory. This is a quick and efficient way to copy the files into a WSAD project working directory.
2. Select the **Turn on tracing** options if you want to see the details of the process.
3. Click the **Deploy EJB Jar** button.

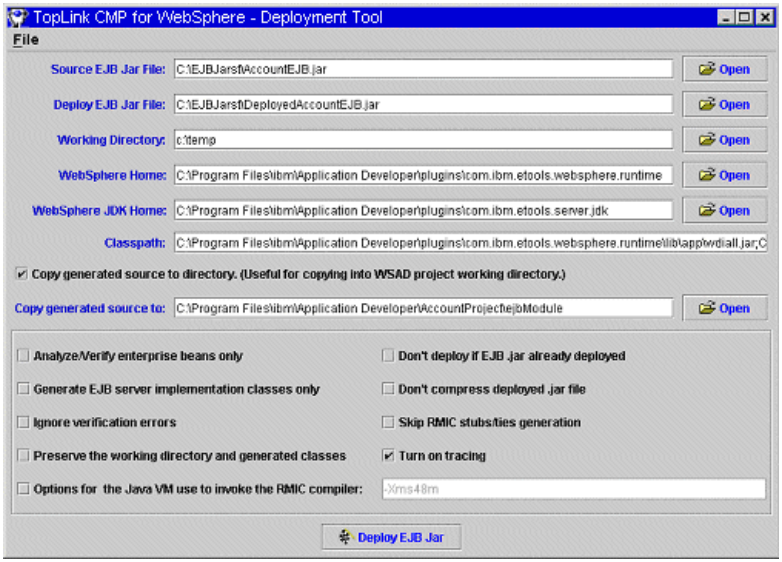
Using Deploy Tool with WebSphere Studio Application Developer (WSAD)

The Deploy tool is completely compatible with the WebSphere Studio Application Developer (WSAD).

To deploy from the Deploy Tool to WSAD

1. Select the EJB Project in WSAD and choose to generate **Deploy and RMIC Code**.
2. Export the EJB Project to an EJB JAR, make sure that TopLink project, `toplink-ejb-jar.xml` file are included in the EJB JAR.
3. Start the TopLink Deploy Tool.
4. Choose the EJB project working directory to allow TopLink to override WSAD deploy code with TopLink deploy code.
5. If the source is copied to a directory other than the WSAD EJB Project folder, manually copy the source files to the WSAD EJB Project under the **ejbModule directory** of the project.
6. Enter appropriate directories in the fields of the Deploy Tool.

Figure 4–2 The Deploy Tool set up for use with WSAD



7. Select **Deploy EJB JAR** to create the deployed EJB JAR.
8. Choose **Rebuild all** from the Project menu to compile the TopLink deploy code to incorporate TopLink CMP.

Troubleshooting The most common error is `NoClassDefFoundError` exception which can be corrected by adding the required resources to the "Classpath" input field. Also "Turn on tracing" option helps to debug error during the generation of deployment code. When an obscure error is shown during the generating stub phase, copy the Java command and run it at the command prompt. This gives a more detailed error message.

Deploying a TopLink-Deployed EJB JAR

To deploy to WebSphere Application Server

1. Start the Administration Server if it is not already started.
2. Start the Administrator's Console and deploy the JAR. For information on deploying the JAR, consult your WebSphere documentation.

Note: When deploying an application containing an entity bean, a data source must be set up and associated with the bean. For information on creating and associating data sources, consult your WebSphere documentation.

It is not necessary to deploy the EJB JAR in WSAD, because deployment is carried using the Deploy Tool (see "Using Deploy Tool with WebSphere Studio Application Developer (WSAD)" on page 4-6).

Starting the entity bean

You can start the bean in either the WebSphere Application Server or in WSAD.

To start the bean in IBM WebSphere Application Server

1. Select the application containing the entity beans.
2. Right click and choose **Start**.

A message dialog will be displayed if the bean started successfully. If an error occurs, consult the Troubleshooting section.

To start the bean in WSAD

1. With WSAD running, right click the EJB project and choose **Run on Server**.
2. Optionally, open the **Console** tab of the Server view to view the status of the process.

Running an EJB Client

After the beans have been deployed, an EJB client can be run to access them. The client can either be a SessionBean or a Java program running outside the server.

The EJB client requires the following bean classes in its CLASSPATH: remote interface, home interface, and primary key classes for all the beans accessed. In addition, if the client is a session bean running on the same server as the entity beans and you want to access the local interfaces of the entity beans, you must also include their local and home interfaces on the CLASSPATH.

To lookup a bean's home interface a JNDI InitialContext must be setup. Setting up the initial context requires that the server's URL be supplied.

5

Defining and Executing Queries

TopLink Container-Managed Persistence provides a feature-rich query framework in which complex database queries can be constructed and executed to retrieve entity beans. When using TopLink Container-Managed Persistence, the developer defines the finder methods on the home interface, but does not need to implement them in the entity bean. TopLink provides this required functionality, and offers a number of strategies for creating and customizing finders. The EJB container and TopLink automatically generate the implementation.

Using Finder Libraries

The general steps required to successfully define a finder method for an entity bean using TopLink query framework are as follows:

1. Define the finder method on the entity bean's home interface (as required by the EJB specification).
2. If required, create an implementation for the query. Some query options require that the query be defined in code on a "helper class", but this is not required for most simple queries.

NAMED finders

A NAMED finder refers to a TopLink query that has been registered with the container under a specific name.

Creating NAMED finders

When using NAMED finders, the "find" method on the home interface must correspond to the name of a TopLink query that has been registered with the container. The query is implemented and then registered with the container within a "TopLink descriptor amendment" method or session amendment class.

Example 5–1 A NAMED finder using a TopLink query named findCustomersInCity

```
 public Enumeration findCustomersInCity(String aCity)throws FinderException,
RemoteException;
```

Defining "named" TopLink queries

Before the `findCustomersInCity` finder shown in this example can work, the corresponding named query `findCustomersInCity` must be defined and made known to TopLink. This can be accomplished several ways, depending on the mechanism used to define the query.

Under EJB QL or SQL When using EJB QL or SQL to define a finder, you can either define the query in the Mapping Workbench in the bean descriptor's **Queries** tab, or add the query to the descriptor in a user defined method.

Under the TopLink expression framework When using the TopLink expression framework the query must be added using a user defined method.

The user defined method can take one of two forms:

- A descriptor amendment method specified on the bean descriptor in the TopLink project in the Mapping Workbench
- The `preLogin` method on a session event listener class. Session event listener classes are specified using the `<event-listener-class>` element in the `toplink-ejb-jar.xml` descriptor.

Example 5–2 Create a ReadAllQuery

```
ExpressionBuilder exp = new ExpressionBuilder();
ReadAllQuery = new ReadAllQuery();
query.setReferenceClass(Customer.class);
query.setSelectionCriteria(exp.get("city").equal(exp.getParameter("city")));
query.addArgument("city");
```

Example 5–3 Define `findCustomersInCity` query in the preLogin method of a session event listener class and specify the session event listener class in the toplink-ejb-jar.xml deployment descriptor.

```
public void preLogin(SessionEvent event) {
// create a query...
event.getSession().getDescriptor(Customer.class).getQueryManager().addQuery("fin
dCustomersInCity", query);
}
```

Example 5–4 Define `findCustomersInCity` query in the amendment method of the descriptor

```
public static void amendment(Descriptor descriptor) {
// create a query...

descriptor.getQueryManager().addQuery("findCustomersInCity", query);
```

For more information on creating a named query, see "Query objects" in Chapter 1 of *TopLink: Using the Foundation Library*.

Using the TopLink expression framework

Define the named query in the amendment method, and add the query to the TopLink descriptor's `QueryManager`.

The named query must be defined based on the following:

- If the return type for the finder method on the home interface is `java.util.Enumeration` or `Collection`, then the query object defined must be a `oracle.toplink.QueryFramework.ReadAllQuery`. If the return type is an entity bean's remote interface (that is, only a single entity bean is returned), then the query must be of type `oracle.toplink.QueryFramework.ReadObjectQuery`.
- The reference class must be the bean class against which the finder is querying.
- The arguments defined in the query must exactly match the `parameter names and types` of the corresponding finder element defined on the home interface.

The arguments defined in the query are retrieved via the `builder.getParameter()` call and then are used for comparison purposes in conjunction with various predicates/operators: `equal()`, `like()`, `anyOf()`, and so on. For more information, see *TopLink: Using the Foundation Library*.

Example 5–5 Using the TopLink expression framework

```
public static void addCustomerFinders(Descriptor descriptor) {
/*Enumeration findCustomersInCity(String aCity) Since this finder returns an
Enumeration, the type of the query is ReadAllQuery.The finder is a "NAMED"
finder. It's implementation is a ReadAllQuery that is registered with the
QueryManager.
*
*/
//1 the query is defined
ReadAllQuery query1 = new ReadAllQuery();
query1.setName("findCustomersInCity");
query1.addArgument("aCity");
query1.setReferenceClass(CustomerBean.class);
//2 an expression is used
ExpressionBuilder builder = new ExpressionBuilder();
query1.setSelectionCriteria
builder.get("city").like(builder.getParameter("aCity";
//3 An option at this point would be to set any desired options on the query,
e.g., queryl.refreshIdentityMapResult();
//4 Finally, the query is registered with the querymanager.
descriptor.getQueryManager().addQuery("findCustomersInCity",query1);
}
```

Using the generic NAMED finder

Alternatively you can use a named query without providing the matching implementation on the home interface if you use the generic NAMED finder provided by TopLink for IBM WebSphere Foundation Library. This finder takes the name of the named query and a vector of arguments as parameters.

Example 5–6 The generic NAMED finder

```
public Enumeration findAllByNamedQuery(String queryName, Vector arguments)
throws RemoteException, FinderException;
```

CALL finders

CALL finders allow dynamic creation of queries. These dynamic queries are generated at runtime instead of at deployment time. When using a CALL finder, a TopLink `SQLCall` or `StoredProcedureCall` is passed as a parameter to a finder that returns an `Enumeration`.

Creating CALL finders

The implementation for CALL finders is supplied by TopLink for IBM WebSphere. To make this available to your bean you will have to add the following finder definition to the home interface of your bean.

Example 5–7 A CALL query

```
public Enumeration findAll(Call call) throws RemoteException,
FinderException;
```

Executing a CALL finder

When using a CALL finder, the call is created on the client using the TopLink interface `oracle.toplink.queryFramework.Call`. this call has three implementors: EJBQLCall, `SQLCall` and `StoredProcedureCall`. Refer to the *TopLink: Using the Foundation Library* manual for details on creating instances of these classes.

Example 5–8 Executing an CALL finder (select statement)

```
try {
    SQLCall call = new SQLCall();
    call.setSQLString("SELECT * FROM EMPLOYEE");
    Enumeration employees = getEmployeeHome().findAll(call);
}
```

Example 5–9 Executing an CALL finder (stored procedure)

```
try {
    StoredProcedureCall call = new StoredProcedureCall();
    call.setProcedureName("READ_ALL_EMPLOYEES");
    Enumeration employees = getEmployeeHome().findAll(call);
```

EXPRESSION finders

EXPRESSION finders allow dynamic creation of queries. These dynamic queries are generated at runtime instead of at deployment time. When using an EXPRESSION finder, a TopLink `Expression` is passed as a parameter to a finder that returns an `Enumeration`.

Creating EXPRESSION finders

The implementation for EXPRESSION finders is supplied by TopLink for IBM WebSphere Foundation Library. To make this available to your bean, you will have to add the following finder definition to the home interface of your bean.

Example 5–10 An EXPRESSION query

```
public Enumeration findAll(Expression expression) throws RemoteException,
FinderException;
```

Executing an EXPRESSION finder

When using an EXPRESSION finder, the query is created on the client.

Example 5–11 Executing an EXPRESSION finder

```
try {
    Expression expression = new
    ExpressionBuilder().get("firstName").like("J%");
    Enumeration employees =
    getEmployeeHome().findAll(expression);
}
```

EJBQL finders

EJBQL is the standard query language defined in the EJB 2.0 specification and is available for use in TopLink with EJB 1.1 beans. EJBQL finders enable a specific EJBQL string to be specified as the implementation of the query.

Advantages EJBQL offers several advantages in that it:

- is the EJB 2.0 standard for queries
- can be used for most queries
- can be used in dependent object queries

Disadvantages Some complex queries may be difficult to define using EJBQL.

Creating an EJBQL finder

To create an EJBQL finder

1. Declare the finder on the Home interface.
2. Start the Mapping Workbench.
3. Go to the **Queries** > **Finders** > **Named Queries** tab for the bean.
4. Add a finder and give it the same name as the finder you declared on your bean's home, and add any required parameters.
5. Select and configure the finder.

Following is an example of a simple EJBQL query that takes one parameter. In this example, the question mark (“?”) in '`?name`' is used to bind the argument name within the EJBQL string.

```
SELECT OBJECT(employee) FROM Employee employee WHERE (employee.name =?name)
```

READALL finders

READALL finders allow dynamic creation of queries. These dynamic queries are generated at runtime instead of at deployment time. Using a READALL finder, a TopLink `ReadAllQuery` is passed as a parameter to a finder that returns an `Enumeration`.

Creating READALL finders

The implementation for READALL finders is supplied by TopLink for IBM WebSphere Foundation Library. To make this available to your bean, you will have to add the following finder definition to the home interface of your bean.

Example 5–12 A READALL query

```
public Enumeration findAll(ReadAllQuery query) throws RemoteException,
FinderException;
```

Executing a READALL finder

When using a READALL finder, the query is created on the client.

Example 5–13 Executing a READALL finder

```
try {
    ReadAllQuery query = new ReadAllQuery(Employee.class);
    query.addJoinedAttribute("address");
    Enumeration employees = getEmployeeHome().findAll(query);
}
```

Advanced finder options

There are a number of options that can be used by the experienced TopLink developer. These options should only be used when the developer has a complete understanding of the consequences of making changes to them.

Caching options

Various configurations can be applied to the underlying query to achieve the correct caching behavior for the application. There are several ways to control the caching options for queries.

For most queries, caching options can be set in the Mapping Workbench see "Caching objects" in Chapter 4 of the *Mapping Workbench Reference Guide*). For finders whose queries are manually created (`findOneByQuery`, `findManyByQuery`), caching options must be applied manually using TopLink for Java APIs.

The caching options can be set on a per-finder basis. The valid values are:

ConformResultsInUnitOfWork (default) For finders returning a single result and finders returning a collection, the 'UnitOfWork' cache for the current JTS UserTransaction is queried. The finder's results will conform to uncommitted new objects, deleted objects and changed objects.

CheckCacheByExactPrimaryKey If a finder returning a single object involves an expression that contains the primary key and only the primary key, the cache is checked.

CheckCacheByPrimaryKey If a finder returning a single object involves an expression that contains the primary key, a cache hit can still be obtained through processing the expression against the object in the cache.

CheckCacheThenDatabase A finder returning a single object queries the cache completely before resorting to accessing the database.

CheckCacheOnly For finders returning a single object and finders returning a collection, only the cache is checked; the database is not accessed.

DoNotCheckCache For finders returning a single object and finders returning a collection, the cache is not checked.

For more information about TopLink queries as well as the TopLink UnitOfWork and how it integrates with JTS, see "Chapter 1: Database Sessions" in *TopLink: Using the Foundation Library.*

Disabling caching of returned finder results

By default, TopLink adds to the cache all returned objects whose primary keys are not currently in the cache. This can be disabled if the client knows that the set of returned objects is very large and wants to avoid the expense of storing these objects. This option is not configurable through the deployment descriptors, but can be configured for queries using `dontMaintainCache()` on the TopLink query API:

```
...
ExpressionBuilder bldr = new ExpressionBuilder();
ReadAllQuery raq = new ReadAllQuery();
raq.setReferenceClass(ProjectBean.class);
raq.dontMaintainCache();
raq.addArgument("projectName");
raq.setSelectionCriteria(bldr.get("name").
like(bldr.getParameter("projectName")));
...
```

Refreshing finder results

A finder may return information from the database for an object whose primary key is already in the cache. When set to true, the refresh cache option in the Mapping Workbench indicates that the object's non-primary key attributes are refreshed with the returned information. This occurs on `findByPrimaryKey` finders as well as all EXPRESSION and SQL finders for that bean when set at the bean attributes level.

When refreshing is enabled, the `refreshIdentityMayResult()` method is invoked on the query. This is configured to automatically cascade private parts. If behavior other than private object cascading is desired, use a dynamic finder.

Caution: When issuing refreshing finders while in user transactions, refreshing the object may cause changes already made to that object to be lost.

In the case where an `OptimisticLock` field is in use, the refresh cache option can be used in conjunction with a global setting for a bean so that the non-primary key attributes are refreshed only if the version of the object in the database is newer than the version in the cache. In the amendment method for a bean, the method `onlyRefreshCacheIfNewerVersion()` is called on the passed in TopLink Descriptor argument.

The following example illustrates how to set `onlyRefreshCacheIfNewerVersion()` option for a bean:

```
public static void addOrderFinders(Descriptor descriptor) {
    ...
    descriptor.onlyRefreshCacheIfNewerVersion();
}
```

For finders that have no refresh cache setting, the `onlyRefreshCacheIfNewerVersion()` method has no effect.

6

Run time considerations

This chapter discusses some of the relevant run-time issues surrounding writing an application that uses TopLink Container-Managed Persistence in the IBM WebSphere Server container. Other facets of the run-time execution that relate to EJB's and the IBM WebSphere Server are beyond the scope of this document and should be reviewed in the EJB specification and/or the IBM WebSphere Server documentation.

Transaction support

Entity beans that use container-managed persistence may participate in transactions that are either *client-demarcated* or *container-demarcated*.

Clients of entity beans may directly set up transaction boundaries using the `javax.transaction.UserTransaction` interface. Invocations on entity beans are automatically wrapped in transactions that are initiated by the container based upon the *transaction attributes* supplied in the EJB deployment descriptor.

For more information on how to use transactions with EJBs, consult the EJB specification and the IBM WebSphere Server documentation. The following sections describe briefly how TopLink participates in EJB transactions.

TopLink within the IBM WebSphere Server

Within the IBM WebSphere Server, TopLink provides a persistence layer for entity beans. While the IBM WebSphere Server controls all aspects of transaction management, the TopLink layer is synchronized with the IBM WebSphere transaction service so that updates to the database are carried out at the appropriate times.

When updates occur

In general, TopLink does not issue updates to the underlying data store until the transaction that the enterprise beans are active in begins its two-stage commit process. This allows for:

- SQL optimizations to ensure that only changed data is written out to the data store
- Proper ordering of updates to allow for database constraints

Valid transactional states

All modifications to persistent beans and objects should be carried out in the context of a transaction. The transaction may either be client-controlled or container-controlled.

The TopLink container does not support modifying beans through their remote interface when no transaction is active. In this case, TopLink does not write out any changes to the data. Modifying entity beans without a transaction leads to an inconsistent state, potentially corrupting the values in the TopLink cache. Transactional attributes MUST be properly specified in the bean deployment descriptors, to ensure that data is not corrupted.

Although it is not valid to modify entity beans through their remote interface without a transaction, in the current release it is permitted to invoke methods on EJB homes that change the state in the underlying database. Invocation of removes and creates that are invoked against homes in the absence of a transaction are permitted.

The following table shows various combinations of container transaction attributes and client transaction behavior. For each case, it is shown whether or not a transaction will be active. For those situations that read "no transaction is active," no modifications to entity beans should be carried out.

Table 6–1 Container transaction behavior as a function of transaction attribute and UserTransaction existence

Transaction attribute	Client transaction exists	No client transaction exists
NotSupported	No transaction is active	No transaction is active
Supports	Transaction is active	No transaction is active
Required	Transaction is active	Transaction is active
RequiresNew	Transaction is active	Transaction is active

Table 6–1 Container transaction behavior as a function of transaction attribute and UserTransaction existence (Cont.)

Transaction attribute	Client transaction exists	No client transaction exists
Mandatory	Transaction is active	Exception is raised
Never	Exception is raised	No transaction is active

Situations described above for which "no transaction is active" should be avoided if entities are to be modified. Bean developers should be particularly careful of using the `Supports` transaction attribute, because it leads to a non-transactional state whenever the client does not explicitly provide a transaction.

Maintaining bidirectional relationships

When one-to-many or many-to-many mappings are bi-directional then the back-pointers must be correctly maintained as the relationships change. When the relationship is between an entity bean and a Java object, or when the application is built to the EJB 1.1 specification (as is the case when using IBM WebSphere Application Server), the relationship must be maintained manually.

One-to-Many relationship

In a one-to-many mapping, an `EmployeeBean` might have a number of dependent `phoneNumbers`. When a `phoneNumber` is added to an employee record, the `phoneNumber`'s back-pointer to its owner (the employee) must also be set.

Example 6–1 Setting the back-pointer in the entity bean

Maintaining a one-to-many relationship in the entity bean involves getting the local object reference from the context of the EmployeeBean, then updating the back-pointer. The following code illustrates this technique:

```
// obtain owner and phoneNumber
owner = empHome.findByPrimaryKey(ownerId);
phoneNumber = new PhoneNumber("cell", "613", "5551212");
// add phoneNumber to the phoneNumbers of the owner
owner.addPhoneNumber(phoneNumber);
```

The Employee's addPhoneNumber() method maintains the relationship as follows:

```
public void addPhoneNumber(PhoneNumber newPhoneNumber) {
    //get, then set the back pointer to the owner
    Employee owner = (Employee)this.getEntityContext()
    .getEJBLocalObject();
    newPhoneNumber.setOwner(owner);
    //add new phone
    getPhoneNumbers().add(newPhoneNumber);
}
```

Managing dependent objects

The EJB 1.1 specification recommends that entity beans be modeled such that all dependent objects are regular Java objects and not entity beans. If a dependent or privately owned object is to be exposed to the client application it must be serializable (it must implement the `java.io.Serializable interface`) so that it may be sent over to the client and back to the server.

Serializing Java objects between client and server

Recall that entity beans are remote objects. This results in a "pass-by-reference" situation when entity beans are referenced remotely. When an entity bean is returned to the client, a remote reference to the bean is returned.

Regular Java objects are not remote objects like entity beans are. Instead of a "pass-by-reference" situation, when regular Java objects are referenced remotely they are "passed-by-value" and serialized (copied) from the remote machine that they were originally on.

Managing collections of EJBObjects

Collections typically use the `equals()` method to compare objects. However, in the case of a Java object that contains a collection of entities, the `EJBObjects` do not respond as expected to the `equals()` method. In this case, the `isIdentical()` method should be used instead. Consequently, you cannot expect the standard collection methods such as `remove()` or `contains()` to work properly when applied to a collection of `EJBObjects`.

Note: This issue does not arise in the case of an entity containing a collection of entities, because a special EJB 2.0 container collection is used which handles equality appropriately.

Several options are available when dealing with collections of `EJBObjects`. One option is to create a helper class to assist with collection-type operations. An example of such a helper is provided in the distribution named `EJBCollectionHelper`:

```
public void removeOwner(Employee previousOwner){
EJBCollectionHelper.remove(previousOwner, getOwners());
}
```

The implementation of remove() and indexOf() in EJBCollectionHelper is shown in the next example:

```
public static boolean remove(javax.ejb.EJBObject ejbObject, Vector vector) {
    int index = -1;
    index = indexOf(ejbObject, vector);
    // indexOf returns -1 if the element is not found.
    if(index == -1){
        return false;
    }
    try{
            vector.removeElementAt(index);
    } catch(ArrayIndexOutOfBoundsException badIndex){
        return false;
    }
    return true;
}
public static int indexOf(javax.ejb.EJBObject ejbObject, Vector vector) {
    Enumeration elements = vector.elements();
    boolean found = false;
    int index = 0;
    javax.ejb.EJBObject current = null;
    while(elements.hasMoreElements()){
        try{
            current = (javax.ejb.EJBObject)
            elements.nextElement();
            if(ejbObject.isIdentical(current)){
                found = true;
                break;
            }
        }catch(ClassCastException wrongTypeOfElement){
```

```
            . . .
        }catch (java.rmi.RemoteException otherError){
            . . .
        }
        index++; //increment index counter
    }
    if(found){
        return index;
    } else{
        return -1;
    }
}
```

If JDK 1.2 is used, a special Collection class could be created that uses `isIdentical()` instead of **`equals()`** for all its comparison operations. For `isIdentical()` to function correctly, the `equals()` method must be properly defined for the primary key class.

7

Customization

With container-managed persistence (CMP), many aspects of persistence are handled transparently by the EJB "container". Other properties may be configured, as required, in the bean deployment descriptors (see "Configuring entity bean deployment descriptors" on page 4-2). The intent is to minimize the amount of persistence code that the EJB developer has to write.

However, there are cases where a bean developer or deployer wants to take advantage of advanced features that require additional customization and configuration of bean deployment.

TopLink Container-Managed Persistence provides a number of entry points for advanced customization of mappings, logins, and other aspects of persistence. These can be used to take advantage of advanced TopLink features, JDBC driver features, or to gain "low-level" access to TopLink for Java APIs that are normally masked in the container-managed persistence layer.

Note: For basic information about TopLink descriptors and mappings, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

Customizing TopLink descriptors and mappings

TopLink projects and descriptors are normally created using the TopLink Mapping Workbench. The output of the TopLink Mapping Workbench tool is an XML file that contains all of the mapping information required to store beans and persistent objects in the database.

Some customizations available to the TopLink descriptors that make up the project cannot be configured using the Mapping Workbench. In these situations, customize the mapping information by specifying an *amendment method* to be run at deployment

time. Each TopLink descriptor can have an amendment method.The TopLink descriptor can also be modified through a Session amendment class because the TopLink descriptors are available through the session. For more information, see "Customizing TopLink descriptors with amendment methods" on page 7-4.

Alternatively, the TopLink Mapping Workbench can be bypassed entirely, and create all the mappings directly in Java code. With this approach, any customizations can be made directly in the source code.

Creating projects and TopLink descriptors in Java

Creating mappings and TopLink descriptors directly in Java code provides access to features that are not available in TopLink Mapping Workbench.

To define a project using Java code:

1. Implement a project class that extends the `oracle.toplink.sessions.Project` class.
2. Compile the project class.
3. Edit the `toplink-ejb-jar.xml` deployment descriptor so that the value for the `<project-class>` element is the fully-qualified Project class name. For more about creating project classes, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

> **Note:** The TopLink Mapping Workbench can be used to create a Java Project class from an existing project which can be used as a starting point for a custom project class. See the *Oracle9iAS TopLink Mapping Workbench Reference Guide* for more information.
>
> Also note that the TopLink Mapping Workbench has an **Export Project to Java Source...** option which can be used as starting point for coding the project class manually.

The following example illustrates how TopLink projects can be specified in code.

```
/**
* The class EmployeeProject is an example of a TopLink project defined in Java
code. The individual parts of the project - the Login and the descriptors, are
built inside of methods that are called by the constructor. Note that
EmployeeProject extends the class oracle.toplink.sessions.Project.
*/
public class EmployeeProject extends oracle.toplink.sessions.Project{
```

```
/**
* Supply a zero argument constructor that initializes all aspects of the
project. Make sure that the login and all the descriptors are initialized and
added to the project.
*/
public EmployeeProject(){
applyPROJECT();
applyLOGIN();
buildAddressDescriptor();
buildEmployeeDescriptor();
// other methods to build all descriptors for the project
/**
* Project-level properties, such as the name of the project, should be specified
here.
*/
protected void applyPROJECT(){
setName("Employee");
}
protected void applyLOGIN()
{
oracle.toplink.sessions.DatabaseLogin login = new
oracle.toplink.sessions.DatabaseLogin();

// use platform appropriate for underlying database
login.setPlatformClassName( "oracle.toplink.internal.databaseaccess.
OraclePlatform");

// if no sequencing is used, setLogin() will suffice
setLoginAndApplySequenceProperties(login);
}

/**
* Descriptors are built by defining table info, setting properties (caching,
etc.) and by adding mappings to the descriptor.
*/
protected void buildEmployeeDescriptor() {
oracle.toplink.publicinterface.Descriptor descriptor =
new oracle.toplink.publicinterface.Descriptor();

// SECTION: DESCRIPTOR
// specify the class to be made persistent
descriptor.setJavaClass(examples.ejb.cmp11.advanced.EmployeeBean.class);
```

```
// specify the tables to be used and primary key
Vector tables = new Vector();
tables.addElement("EJB_EMPLOYEE");
descriptor.setTableNames(tables);
descriptor.addPrimaryKeyFieldName("EJB_EMPLOYEE.EMP_ID");

// SECTION: PROPERTIES
descriptor.setIdentityMapClass(
oracle.toplink.internal.identitymaps. FullIdentityMap.class);
descriptor.setExistenceChecking("Check cache");
descriptor.setIdentityMapSize(100);

// SECTION: COPY POLICY
descriptor.createCopyPolicy("constructor");

// SECTION: INSTANTIATION POLICY
descriptor.createInstantiationPolicy("constructor");

// SECTION: DIRECTTOFIELDMAPPING
oracle.toplink.mappings.DirectToFieldMapping firstNameMapping = new
oracle.toplink.mappings .DirectToFieldMapping();
firstNameMapping.setAttributeName("firstName");
firstNameMapping.setIsReadOnly(false);
firstNameMapping.setFieldName("EJB_EMPLOYEE.F_NAME");
descriptor.addMapping(firstNameMapping);

// … Additional mappings are added to the descriptor using the addMapping()
method.
},}
```

After the TopLink project is written and compiled, it can be used in deployment. You can specify the project class to be used instead of a project file by filling in the `project-class` element in the `toplink-ejb-jar.xml` deployment descriptors for your entity beans.

Customizing TopLink descriptors with amendment methods

The TopLink descriptor of any persistent class can be modified when the descriptor is first instantiated. For container-managed persistence, this happens when the entity beans are deployed into the EJB server.

Amendment methods are static methods that are run at deployment time and allow for arbitrary descriptor customization code to be run.

For more information on amendment methods, see the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

Working with TopLink ServerSession and Login

TopLink interacts with databases using two key components:

- The `ServerSession` is a TopLink component that interacts with the underlying database on behalf of the application.
- The `DatabaseLogin` contains connection information and settings that are specific to the underlying database.

Understanding ServerSession

In TopLink container-managed persistence support, the `ServerSession` is normally hidden from the EJB developer because interaction with the database is performed transparently by the EJB container (via TopLink). The `ServerSession` is still present "behind-the-scenes", but plays a lesser role in its direct interaction with the EJB application.

The `ServerSession` handles all aspects of persistence, such as caching, reading and writing.

Understanding DatabaseLogin

Databases typically require a valid username and password to login successfully. In a TopLink application, this login information is stored in the `DatabaseLogin` class. All sessions must have a valid `DatabaseLogin` instance before logging in to the database.

For more information on `DatabaseLogin`, see "Database Sessions" in the *Oracle9iAS TopLink Foundation Library Guide*.

Customizing ServerSession and DatabaseLogin

A session amendment class can be used to configure the ServerSession and DatabaseLogin in ways not available through the deployment descriptor file.

The `ServerSession` and `DatabaseLogin` may need to be customized for any of the following reasons:

- You need to specify special settings for the JDBC driver, such as to use *parameter binding* or to use a different data conversion routine to work with an incompatible driver
- You wish to directly access regular TopLink for Java features, such as database connections or caching

- You want to define custom finder queries on one or more TopLink descriptors.

Other settings that can be applied to the `ServerSession` and `DatabaseLogin` are:

- Native SQL support — required if your JDBC bridge does not support the JDBC standard SQL syntax
- Binding and parameterized SQL — these options determine whether values are inlined directly into the generated SQL or are parameterized
- Batch writing — allows groups of insert/update/delete statements to be sent to the database in a single batch
- Optimizing data conversion

Additional configuration changes

You can register a session listener class that extends `oracle.toplink.sessions.SessionEventAdaptor` to listen for various session events, such as `pre_login`, `post_commit_unit_of_work`, and so on. The listener is registered to the TopLink session by defining the `<event_listener_class>` tag in the `toplink-ejb-jar.xml` file. For example:

```
<session>
    <event_listener_class>
    oracle.toplink.ejb.cmp.demos.sessionlistener
    </event_listener_class>
</session>
```

8

The Single Bean Example Application

TopLink includes several TopLink EJB 1.1 CMP example applications for WebSphere in the `\examples` directory:

- Single Bean
- Relationships
- Advanced

See `<INSTALL_DIR>/doc/demos.html` for links to all the examples and details on configuring the examples for WebSphere Application Server.

This chapter details how the Single Bean example is built. Although the Single Bean example involves persisting just one entity bean, the information provided on development, configuration and deployment is the same as is needed for more complex multi-bean applications.

Understanding the Single Bean example

The Single Bean example application shows how a single bean can be made persistent using TopLink Container-Managed Persistence container managed persistence support. This example illustrates simple direct-to-field mappings and introduces the basic steps required to deploy a bean. The example consists of an entity bean called AccountBean.

The example is configured to on the IBM WebSphere Application Server.

The Single Bean example application demonstrates:

- The use of an entity bean in an application
- That persistence-related code is not required in the entity bean

- That implementation of ejbFind methods is not required in the entity bean (in the example, query logic for each finder method included in the home interface is defined using the Mapping Workbench or in an amendment method)
- The use of the TopLink deployment XML file, which contains bean-to-database mapping meta information
- How TopLink transparently manages persistence for beans when they are being created, updated, removed, and queried

The Client application performs the following steps:

- Creates a simple AccountBean instance, makes a deposit and a withdrawal from the account
- Populates the database with several account bean instances
- Finds all accounts in the database whose balances are greater than the provided sum
- Finds an account in the database whose owner matches a provided search string
- Removes all accounts created

The Object model

The Single Bean example provides a simplified view of the standard "bank account" example, and shows how a single class can be modeled as an entity bean and made persistent using TopLink.

The interface

`examples.ejb.cmp11.singlebean.Account` provides the public interface for the bean. It extends the `javax.ejb.EJBObject` interface, and contains all of the business methods that are accessible to clients of the entity. This includes getters and setters for the instance data, as well as `deposit()` and `withdraw()` methods.

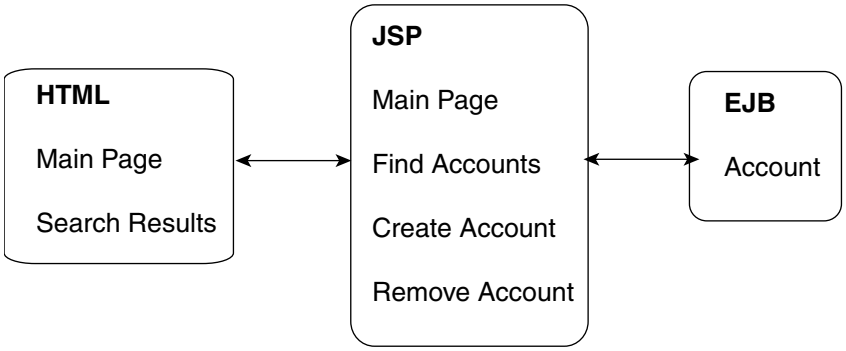
The class

`examples.ejb.cmp11.singlebean.AccountBean` provides the actual bean implementation for the bank account. It has methods corresponding to the methods on the remote interface, as well as the methods required by the `javax.ejb.EntityBean` interface, which it extends. The AccountBean's fields include `accountId (String)`, `balance (double)`, and `owner (String)`.

The home interface

`examples.ejb.cmp11.singlebean.AccountHome` provides the "home" interface of the bean. It extends the `javax.ejb.EJBHome` interface, and defines the required create, remove, and finder methods.

Figure 8–1 Structure of the Single Bean Example



Database schema

The Account data is stored in a single table.

Table 8–1 The EJB11_ACCOUNT table

Column Name	Column Type	Details
ACCOUNT_ID	VARCHAR	primary key
BALANCE	DOUBLE / NUMERIC	balance in account
OWNER	VARCHAR	owner's name

Entity Development

A deployable component is typically developed as follows:

- Create the interfaces.
- Create and implement the bean classes.
- Create the deployment descriptors.
- Map the entities to the database.

- Generate code for deployment.
- Deploy the EAR file.
- Run the client.

Create the interfaces

Each entity can contain either a home or remote interface, or both. These interfaces dictate how the bean is used by other components of the application. The Account interfaces required for the Single Bean example are located in the `was/examples/ejb/cmp11/singlebean` example directory.

Create and implement the bean classes

Define instance variables for each CMP field and corresponding get and set methods. Also implement business methods on the bean. The AccountBean class required for the Single Bean example is located in the `was/examples/ejb/cmp11/single bean` example directory.

Create the deployment descriptors

A JAR file requires an `ejb-jar.xml` and a `toplink-ejb-jar.xml`. The EAR file requires other descriptors, as do client application JAR files. The WebSphere AAT or other tools can be used to generate all but the `toplink-ejb-jar.xml`. The developer must manually create this TopLink deployment descriptor.

The TopLink deployment descriptor: toplink-ejb-jar.xml

The TopLink deployment descriptor is included in the EJB JAR in the same META-INF directory as the `ejb-jar.xml` and the ibm extension files. This descriptor provides the information that TopLink needs to deploy the entities in the EJB JAR.

Because the entities deployed in a EJB JAR are all encompassed by a TopLink project, the deployment JAR file is associated with exactly one project. This project is in turn associated with exactly one TopLink session (as implied by the single session element in the descriptor).

The elements that have been modified for the Single Bean example in the `toplink-ejb-jar.xml` file are:

<name> A session name (unique among all deployed JARs) that is used as a key for the deployed TopLink project (or the JAR that contains the project).

<project-xml> project deployment XML file that can be stored either in the deployable JAR file at the root directory or left on the file system.

<session-type> The session type should always be set to <server-session/>.

<platform-class> The class specified controls the format of the SQL generated and other database specific behavior.

> **Note:** You can use a `<project-class>` element rather than a `<project-xml>` if you choose. With the `<project-class>` element, specify the fully-qualified name of the TopLink project class. This class should be included in the deployable JAR file. The project class can either be generated by the Mapping Workbench or written manually.

<uses-external-connection-pool> and <uses-external-transaction-controller> For TopLink to participate in WebSphere JTS transactions these should be both set to true.

<external-transaction-controller-class> This is the TopLink server-specific JTS controller class required when using external transaction control. For WebSphere 4.0 use `oracle.toplink.jts.was.JTSExternalTransactionController_4_0`.

<enable-logging> When set to true, TopLink will print logging information for several of its operations. This is very useful for debugging.

<logging-options> Options for different levels of TopLink logging.

For more information, see the toplink-was-ejb-jar_903.dtd.

Map the entities to the database

This section describes the steps required to create the Account project using the Mapping Workbench.

For more information about creating projects using the Mapping Workbench, consult the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

This section assumes you have already read and completed the introductory tutorials in *Oracle9iAS TopLink Tutorials*, which offers an introduction to the fundamental concepts of the Mapping Workbench.

Creating a TopLink project

A TopLink project defines how the entity beans are persisted to the database. The Mapping Workbench enables you to easily build a TopLink project. The project is specified in the `toplink-ejb-jar.xml` in the `<project-class>` or `<project-xml>` element and used at run time to persist the beans.

> **Note:** The complete Account project is available in the `was/examples/ejb/cmp11/singlebean/mw` directory and can be opened and examined with the Mapping Workbench.

To create a TopLink project:

1. Click **File** > **New Project** to start a new project.
2. In the General tab, set the Persistence Type to 1.1 CMP.
3. Optionally, specify an `ejb-jar.xml` file to use for the project. For EJB 1.1 projects it is not required or even desirable to specify and `ejb-jar.xml` file here since under the EJB 1.1. specification the `ejb-jar.xml file` contains no mapping information that is not readily available in the bean classes.

   For the Mapping Workbench to be able to read and update the `ejb-jar.xml`, it must have the EJB 2.0 DTD its DOCTYPE element. See "Working with the ejb-jar.xml file" in the *Mapping Workbench Reference Guide* for details on working with the `ejb-jar.xml` file.
4. In the General tab, specify a project classpath. The project classpath should contain the classes to be added to the project and interfaces associated with those classes. Classes to be added to the project include bean classes and referenced classes. Bean interfaces do not have to be added to the project, but must appear in the project classpath.
5. To add the beans to the project, click **Selected** > **Add/Refresh Classes**.

   > **Note:** Bean classes must to be added to the project at this point (for example, the Single Bean example requires the AccountBean class); however, referenced classes are not required.

6. To specify the beans classes as bean descriptors, click **Selected** > **Descriptor Type** > **EJB Descriptor**.

7. Create database tables. The Single Bean example application uses an EJB11_ACCOUNT table to persist the bean. Ensure that the ACCOUNT_ID is the primary key in the EJB_ACCOUNT table.

   The table can either be imported from the database or created in the Mapping Workbench. For more information on working with tables, see "Working with database tables" in the *Oracle9iAS TopLink Mapping Workbench Reference Guide*.

8. To associate AccountBean with a table, select the AccountBean and set the EJB_ACCOUNT table as the associated table in the Descriptor Info tab.

9. Map the CMP fields. The AccountBean has three CMP fields to be mapped using direct-to-field mappings: accountId, balance, and owner. Map them to their corresponding database fields in the EJB11_ACCOUNT table.

10. Create queries for finders. The AccountHome defines two finders that must be defined: `findByOwner` and `findLargeAccounts`. The Queries tab of the AccountBean descriptor is used to define a query for each finder on the AccountHome. Each query requires:

    - The name of the query
    - The Type of query depending on whether the query returns a single bean or a collection of beans
    - The query format (one of SQL, EJBQL, or TopLink's expression framework)
    - The Query String used for SQL or EJBQL query strings. If TopLink's expression framework is used then the query must be defined in a descriptor after-load method
    - Parameters and Options

    Note: The finder findByPrimaryKey is implemented by TopLink and does not require a user implementation.

11. Export a project to be used at runtime. The project can be written out as a Java class which has to be compiled and included with the deployment JAR or an XML file. In the `toplink-ejb-jar.xml` file, use either the `<project-class>` or `<project-xml>` depending on which export method was used.

Generate code for deployment

The Single Bean example is packaged into an EAR file, which itself contains the following:

- A deployable EJB 1.1 JAR file containing the interface and abstract bean classes, the classes (RMI stubs and implementation classes) generated by IBM's deploytool and TopLink's deploytool, and the deployment descriptor XML files.
- A client JAR containing the client code.

The TopLink deployment code generation tool must be used instead of WebSphere's to generate the deployable JAR file. The generated code contains callbacks to the TopLink persistence framework which makes CMP possible. TopLink's deploytool takes a non-deployed EJB 1.1 JAR as input, generates the TopLink-specific CMP code and calls the WebSphere deploytool for code generation.

EJB 1.1 JAR files can be created by any of a combination of the following tools:

- VisualCafe 4.5 with WebSphere 4.0 Integration plugin
- WebSphere Application Assembly Tool
- WebSphere Studio Application Developer (WSAD)
- VisualAge for Java (VAJ)
- JDK packaging tools

TopLink's deploytool can be launched from the TopLink CMP Deployment Tool or from the command line.

In Windows, the deploy tool can be opened from the **Start** menu by clicking **Start** > **Programs** > **Oracle9iAS TopLink** > **Tools** > **Deploy Tool for WebSphere Server**. The deploy tool can be used with WebSphere Application Server or with WebSphere Studio Application Developer.

An Ant-based build script is included with the example application, which disassembles an EAR file, calls TopLink code generation on the JAR file, and reassembles the EAR file. A copy of the deployable JAR is placed in the server's `installableApps\` directory.

Deploy the EAR file

You can deploy the EAR file several different ways (see the WebSphere Server documentation for more detailed information on how to deploy a EAR file into the server).

The Single Bean example is configured to run against the local HSQL database. Ensure that the HSQL database server is started before starting the application on the server. Also ensure that the HSQL `toplink.examples.datasource` is configured in the server before installing the application on the server. See the examples configuration documentation in the `examples/` directory for details.

Run the client

Included in the Single Bean directory, is a `runClient.cmd` which can be used to run the client.

A

EJB Architectures Summary

Enterprise JavaBeans present a way to build components as well as a means to make these components exist in a transactional, secure, and distributed environment. However, a single bean represents only one component - and consequently only one part of a complete application. EJB provides developers with flexibility in determining how these components should be made to work together. There are a number of ways in which Enterprise JavaBeans can be made to work together to form a complete enterprise application. TopLink can be integrated into each variety of EJB application architecture to provide both the technology that enables these architectures and the features that add value to them.

This chapter gives an overview of some of the basic design patterns available when using TopLink and TopLink CMP. It is not meant to be prescriptive and neither is it complete. It briefly suggests some of the more useful EJB designs and their suitability to specific applications. Architects and developers may find these sections useful at the early stages of application design. As more experience is acquired the appropriateness of particular patterns will become more obvious, and architectural decisions will be more intuitively reached.

Note: Although discussed in the context of EJB 2.0, the architectures presented in this chapter are still valid in an EJB 1.1 environment. Any time entity beans are mentioned in terms of their local interface, these can be replaced with beans using their remote interface. The only downside to this strategy is the extra overhead involved with remotes.

Introduction to EJB architectures

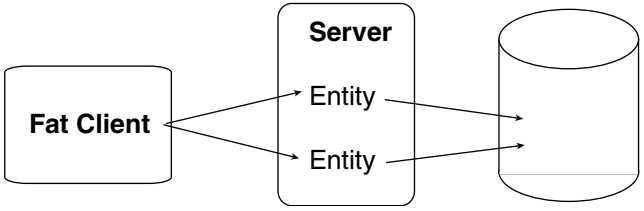
The basic ways in which EJBs can be assembled, or the basic "EJB architectures", can be described in terms of which kinds of beans or J2EE components are used, how client applications access them, and how the underlying "domain objects" are represented. The EJB architectures can be fundamentally divided into three categories: an Entity bean architecture, a Session bean architecture, and a Session and Entity "tiered" approach. Each basic architecture also has variations and refinements and can be decorated with a variety of J2EE components.

The EJB specification does not dictate how enterprise entities are used, but it is clear from the evolution of the specification that certain architectures were assumed to be dominant. Some of the recommended architectures are explained in the J2EE Blueprints (see http://java.sun.com/blueprints). These documents should be reviewed for more information about J2EE and EJB architectures.

Remote Entities

If entities alone are used then they must have remote interfaces that expose all of the client servicing methods. In the absence of Session beans, the client may only access entity state through its remote interface, and may not traverse relationships, except as encapsulated by remote method calls. Only remote references and data may be returned by these calls. Finders may only return remote references as well.

Figure A–1 Remote entities architecture



Clients gain from this approach in that the distribution of the entities is transparent. Clients reference the entities as if they were local, and do not need to worry about location. Entities can exist at different locations without the client even being aware of it.

The converse of transparent distribution is that if clients are not aware of the distributed nature of the entities then they may not be aware of the cost of invoking them. If the entities are "fine-grained" objects then each fine-grained method invocation on them will end up being a remote call. The accumulation of these

remote method invocations could sum up to a potentially serious network or communication latency cost.

If Container transactions are used for each entity operation a separate transaction will end up being initiated for each method invocation. This could introduce excessive and unnecessary transaction management overhead if client-demarcated UserTransactions are not used.

Since Entity beans are intended to be "components" there are more restrictions placed on them than on regular Java objects (e.g. thread-spawning is disallowed). This may impose limits on how they can be used to model certain domain concepts. The limitations should be well understood and compared against the model to ensure that they are not discovered too late in the design phase.

In general, however, this architecture is less desirable than other architecture types. The relationship limitations imposed by the EJB specification are often an impediment to using this approach.

Advantages

- Increased distribution
- Potential for greater location transparency

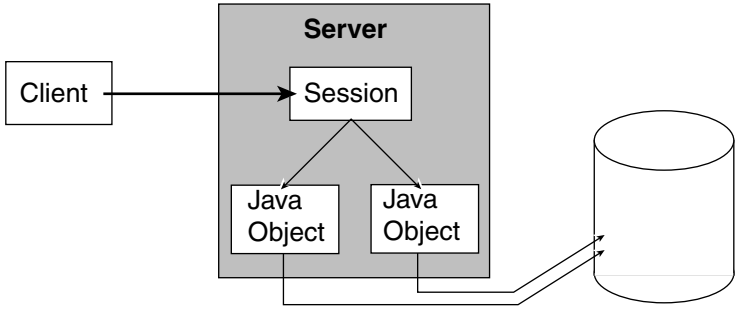
Disadvantages

- Not suitable for fine-grained entities
- Communication overhead for each method call
- Transactional costs of each method call
- Client accesses many EJB interfaces (no single client interface or point of server entry)
- Much of the business logic must reside on the client

Remote Session beans

It is common practice to apply a Session bean layer in front of lightweight objects. This may take the form of the session façade pattern described below, with lightweight entities or other types of persistent objects being managed.

Figure A–2 Remote session beans architecture



Since Session beans do not themselves represent durable objects, often a session façade pattern will be used to converse directly with persistent Java objects, without the use of entities. Persistent data can be modeled using regular TopLink-enabled persistent Java objects that are managed by the Session beans and mapped using TopLink tools. Since few domain objects actually "live" on the client, client applications rarely need to access the domain objects directly, but if regular objects are used then they may be sent to the client if necessary since no such restriction exists for them. The Session beans are used to carry out most of the application logic. Stateful beans are used for those operations for which client-identity is important, while stateless Session beans can be used for "single-shot" operations. All of the EJB benefits of security, transactions and distribution are available through the Session beans.

The exclusive use of Session beans does not allow for overly complex client behavior - all client behavior is limited to services provided by the Session beans. Simple client behavior is a general characteristic of all thin client architectures.

Simplicity and fast client access are clear benefits of this approach. In addition, there is great flexibility in how the domain objects are designed, and how these objects are mapped to the underlying relational database tables.

Advantages

- Location transparency on session interface
- Fine-grained persistent object operations can be "batched" or combined into a single session bean call to reduce communication overhead
- High performance storage/retrieval of persistent objects
- EJB features available through single point of session bean entry to reduce Container overhead

- Can relate persistent objects
- Can pass persistent objects to the client side if necessary

Disadvantages

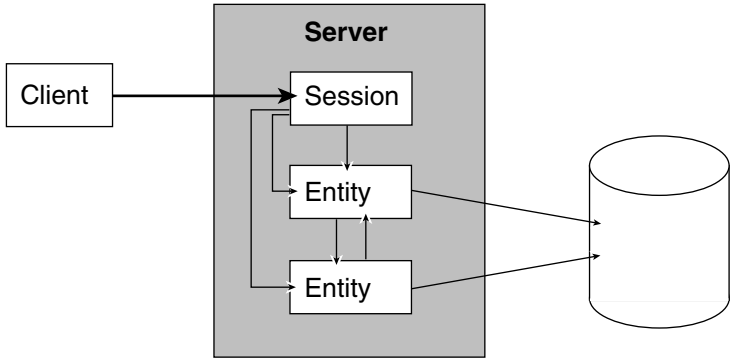
- Fine-grained client calls may require explosion of session interface calls
- Persistent Java objects not included in EJB specification

Session Façade - Combining Session and Entity beans

The majority of systems have relationships between application entities. This being the expected scenario, it is well observed and explained by J2EE designers. More common, however, is to incorporate the domain logic into the session bean itself. This migrates the business logic from the client to the server which provides a number of well-known benefits including ease of maintenance, convenient upgradability, and increased access to server features.

Regardless of whether the session bean simply forwards operations to entities or actually includes the application logic as a façade that fronts the local entities it is a modular approach to remotely accessing server-side objects. It is also likely to be easier to maintain as the J2EE specification moves forward, since session beans tend to experience change to a lesser degree than other components, such as entities. The decision to use a stateful or stateless session bean will likely depend on the amount of business logic incorporated into the session bean.

Figure A–3 Session façade architecture



Advantages

- Location transparency on session interface
- Fine-grained entity operations can be “batched” or combined into a single session bean call to reduce communication and transaction overhead
- Inter-entity method calls are pass-by-reference
- Can maintain entity relationships
- All components described by EJB specification
- Flexibility to create new (local) transactions on specific method calls when required·

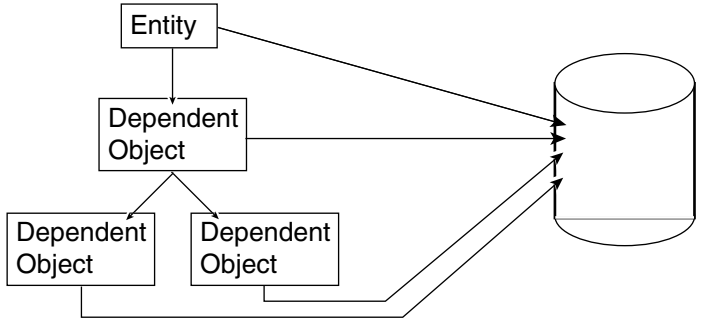
Disadvantages

- Fine-grained client calls may require explosion of session interface calls
- Some Container overhead still incurred on each local entity call

Dependent Java Objects

Because of TopLink's ability to persist regular Java objects without the need for EJB container support these objects offer the most flexibility. They do not incur the entity costs of container management but can be persisted independent of the entity. This means that changes are detected at commit-time, but if nothing changes during the course of an entity update then the object will not be written back, and likewise only the object may be written out if the reverse is true. The dependency upon the entity is still intact, however, as any removal of the entity will automatically propagate to cause the dependent Java object to be removed from persistent storage. Like serializable value objects, they can be transported back and forth between the client and server, allowing for client interactions that refer to the owning bean, but operate on the dependent data. These objects are not supported in the EJB specification, but do offer the benefits of managed, mapped persistence.

Figure A–4 Using dependent Java objects in a system



Conclusion

EJB provides developers with a great deal of infrastructure that makes building enterprise applications easier. This allows developers to build better applications by allowing them to focus on the business logic of their application rather than on distribution, security, and transactions. Even with everything that EJB provides, developing with EJB requires intelligent architectural choices to be made. Although EJB provides much, it also allows for flexibility so that developers can use it to meet their needs.

Regardless of the EJB architecture used, TopLink will support it by providing the right level of control and transparence appropriate to the architecture. The TopLink persistence framework also adds the value required to customize applications to run at their best, and will play an essential role in enabling customers to successfully develop and deploy their enterprise applications.

B

The toplink-ejb-jar DTD

This appendix offers a listing of the `toplink-ejb-jar` document type description (DTD).

DTD listing

```
<!--This is the root element and exists only for XML structure-->
<!ELEMENT toplink-configuration (session* , session-broker*)>

<!--This element used if a session broker must be configured-->
<!ELEMENT session-broker (name , session-name+)>

<!--This is the element that represents the session name-->
<!ELEMENT session-name (#PCDATA)>

<!--This is the node element that describes a particular session for use within
toplink-->
<!ELEMENT session (name , (project-class | project-xml) , session-type , login?
, cache-synchronization-manager? , event-listener-class* , profiler-class? ,
external-transaction-controller-class? , exception-handler-class? ,
connection-pool* , enable-logging? , logging-options?)>

<!--This is the type of session that is being configured-->
<!ELEMENT session-type (server-session | database-session)>

<!ELEMENT server-session EMPTY>

<!ELEMENT database-session EMPTY>

<!--This is the class name that this session will load to provide login and
mapping information-->
<!ELEMENT project-class (#PCDATA)>
```

```
<!--This is the file that contains the project that this session will load to
provide login and mapping information-->
<!ELEMENT project-xml (#PCDATA)>

<!--This is the element that is used if the session will be synchronized with
others-->
<!ELEMENT cache-synchronization-manager (clustering-service , multicast-port? ,
multicast-group-address? , packet-time-to-live? , is-asynchronous? ,
should-remove-connection-on-error? , (jndi-user-name , jndi-password)? ,
jms-topic-connection-factory-name?, jms-topic-name?,
naming-service-initial-context-factory-name?,naming-service-url?)>

<!--This is the name of the clustering service that will be used for connecting
sessions for Cache Synchronization-->
<!ELEMENT clustering-service (#PCDATA)>

<!--This is the IP that the Clustering Service will be listening for new session
anouncements-->
<!ELEMENT multicast-group-address (#PCDATA)>

<!--This is the multicast port the the clustering service will be listening on
for announcements of new sessions-->
<!ELEMENT multicast-port (#PCDATA)>
<!ATTLIST multicast-port  e-dtype NMTOKEN  #FIXED 'number' >

<!--Set to true if synchronization should not wait until all sessions have been
synchronised before returning-->
<!ELEMENT is-asynchronous (#PCDATA)>

<!--Set to true if the connection should be removed from this session if a
communication error occurs-->
<!ELEMENT should-remove-connection-on-error (#PCDATA)>

<!--This is the JNDI name of the Topic Connection Factory that was created for
synchronizing TopLink Sessions-->
<!ELEMENT jms-topic-connection-factory-name (#PCDATA)>

<!--This is the JNDI name of the Topic that was created for synchronizing
TopLink Sessions-->
<!ELEMENT jms-topic-name (#PCDATA)>

<!--This is the name of the initial context factory that will be included in the
Context Properties when creating an initial context for accessing JNDI-->
<!ELEMENT naming-service-initial-context-factory-name (#PCDATA)>
```

```
<!--The URL to the global Namespace for the Synchronization connection.  Usually
the URL of the JNDI service-->
<!ELEMENT naming-service-url (#PCDATA)>

<!--The maximum number of hops a packet will be broadcast-->
<!ELEMENT packet-time-to-live (#PCDATA)>
<!ATTLIST packet-time-to-live  e-dtype NMTOKEN  #FIXED 'number' >

<!--This element used if a user name is required to access the JNDI service in
the case of Cache Synchronization-->
<!ELEMENT jndi-user-name (#PCDATA)>

<!--This element used if a password is required to access the JNDI service in
the case of Cache Synchronization-->
<!ELEMENT jndi-password (#PCDATA)>

<!--This describes one of possibly many event-listeners that can be registered
on the session-->
<!ELEMENT event-listener-class (#PCDATA)>

<!--This element represents the class name of the profiler that will be used by
the session-->
<!ELEMENT profiler-class (#PCDATA)>

<!--This is the class that the session will use as the external transaction
controller-->
<!ELEMENT external-transaction-controller-class (#PCDATA)>

<!--This is the class that the session will use to handle exceptions generated
from within the session-->
<!ELEMENT exception-handler-class (#PCDATA)>

<!--SQL will be logged to the Session writer which, by default, is System.out-->
<!ELEMENT enable-logging (#PCDATA)>

<!--This element used to specify the extra logging options-->
<!ELEMENT logging-options (log-debug? , log-exceptions? ,
log-exception-stacktrace? , print-thread? , print-session? , print-connection? ,
print-date?)>

<!--Debug messages will be logged-->
<!ELEMENT log-debug (#PCDATA)>

<!--exceptions will be logged-->
```

```
<!ELEMENT log-exceptions (#PCDATA)>

<!--exceptions stack traces will be logged when they occur-->
<!ELEMENT log-exception-stacktrace (#PCDATA)>

<!--Each line of the log will contain the connection id-->
<!ELEMENT print-connection (#PCDATA)>

<!--each line of the log will contain the date-->
<!ELEMENT print-date (#PCDATA)>

<!--each line of the log will contain the session id-->
<!ELEMENT print-session (#PCDATA)>

<!--each line of the log will contain the thread id-->
<!ELEMENT print-thread (#PCDATA)>

<!--This the node element that stores the information for the connection
pools-->
<!ELEMENT connection-pool (is-read-connection-pool , name , max-connections? ,
min-connections? , login)>

<!ELEMENT is-read-connection-pool (#PCDATA)>

<!--The max number of connections that will be created in the pool-->
<!ELEMENT max-connections (#PCDATA)>
<!ATTLIST max-connections  e-dtype NMTOKEN  #FIXED 'number' >

<!--The min number of connections that will aways be in the pool-->
<!ELEMENT min-connections (#PCDATA)>
<!ATTLIST min-connections  e-dtype NMTOKEN  #FIXED 'number' >

<!--This is the node element that represents the login for a particular
connection pool.  The read and write connection pools will use the login from
the project-->
<!ELEMENT login (license-path? , driver-class? , (connection-url | datasource)?
, platform-class? , user-name? , password? , uses-native-sequencing? ,
sequence-preallocation-size? , sequence-table? , sequence-name-field? ,
sequence-counter-field? , (should-bind-all-parameters ,
should-cache-all-statements?)? , uses-byte-array-binding? , uses-string-binding?
, uses-streams-for-binding? , should-force-field-names-to-uppercase? ,
should-optimize-data-conversion? , should-trim-strings? , uses-batch-writing? ,
uses-jdbc20-batch-writing? , uses-external-connection-pool? , uses-native-sql? ,
uses-external-transaction-controller? , ((non-jts-connection-url |
non-jts-datasource), uses-sequence-connection-pool?)?)>
```

```
<!--This is the element that represents the platform class name-->
<!ELEMENT platform-class (#PCDATA)>

<!--This is the element that represents the database driver class name-->
<!ELEMENT driver-class (#PCDATA)>

<!--This is the URL that will be used to connect to the database-->
<!ELEMENT connection-url (#PCDATA)>

<!--This is the URL of a datasource that may be used by the session to connect
to the database-->
<!ELEMENT datasource (#PCDATA)>

<!--This element used if a read-only datasource is required for cache
synchronization (usually used within an application server)-->
<!ELEMENT read-only-datasource (#PCDATA)>

<!--This element is used in the login as well as the Cache Synchronization
feature-->
<!ELEMENT user-name (#PCDATA)>

<!--This element is used in the login as well as the Cache Synchronization
feature-->
<!ELEMENT password (#PCDATA)>

<!--Set to true if the login should use native sequencing-->
<!ELEMENT uses-native-sequencing (#PCDATA)>

<!--Sets the sequencing pre-allocation size.  This is the number of sequences
that will be retrieved from the database each time-->
<!ELEMENT sequence-preallocation-size (#PCDATA)>
<!ATTLIST sequence-preallocation-size  e-dtype NMTOKEN  #FIXED 'number' >

<!--The name of the sequence table-->
<!ELEMENT sequence-table (#PCDATA)>

<!--The field within the sequence table the stores that the sequence name-->
<!ELEMENT sequence-name-field (#PCDATA)>

<!--The field within the sequence table that stores the -->
<!ELEMENT sequence-counter-field (#PCDATA)>

<!--Set to true if all queries should bind all parameters-->
<!ELEMENT should-bind-all-parameters (#PCDATA)>
```

```
<!--Set to true if all statements should be cached-->
<!ELEMENT should-cache-all-statements (#PCDATA)>

<!--Set to true if byte arrays should be bound-->
<!ELEMENT uses-byte-array-binding (#PCDATA)>

<!--Set to true if strings should be bound-->
<!ELEMENT uses-string-binding (#PCDATA)>

<!--Set to true if streams should be used when binding attributes-->
<!ELEMENT uses-streams-for-binding (#PCDATA)>

<!--Set to true if field names should be converted to uppercase when generating
SQL-->
<!ELEMENT should-force-field-names-to-uppercase (#PCDATA)>

<!--Set to true if the session should optimize data conversions-->
<!ELEMENT should-optimize-data-conversion (#PCDATA)>
<!--Set to true if the connection should use native SQL-->
<!ELEMENT uses-native-sql (#PCDATA)>

<!--Set to true if trailing white spaces should be removed from strings-->
<!ELEMENT should-trim-strings (#PCDATA)>

<!--Set to true if the connection should batch the statements-->
<!ELEMENT uses-batch-writing (#PCDATA)>

<!--Set tio true if the connection should use jdbc2.0 batch writing-->
<!ELEMENT uses-jdbc20-batch-writing (#PCDATA)>

<!--Set to true if the connection should use an external connection pool-->
<!ELEMENT uses-external-connection-pool (#PCDATA)>

<!--Set to true if the session will be using an external transaction
controller-->
<!ELEMENT uses-external-transaction-controller (#PCDATA)>

<!--Genereic element used to describe a string that represents the name of an
item-->
<!ELEMENT name (#PCDATA)>

<!--This element used if a non-jts connection is required (usually only required
in an Application server when CacheSync is used-->
<!ELEMENT non-jts-connection-url (#PCDATA)>
```

```
<!--This element used if a non-jts connection is required (usually only required
in an Application server when CacheSync is used-->
<!ELEMENT non-jts-datasource (#PCDATA)>
```

Index

A

B

C

D

E

F

G

H

S

T

W