Oracle® Containers for J2EE

Resource Adapter Administrator's Guide

10*g* (10.1.3.1.0)

B28956-01

September 2006

ORACLE®

Oracle Containers for J2EE Resource Adapter Administrator's Guide, 10*g* (10.1.3.1.0)

B28956-01

Primary Author: Joseph Ruzzi

Contributing Authors: Sheryl Maring, Brian Wright, Bonnie Vaughan

Contributors: Jyotsna Laxminarayanan, Anthony Lai, Lelia Yin, Raghav Srinivasan, Frances Zhao, Olivier Caudron, Tug Grall, John Speidel, Vivek Maganty, Tom Beerbower, Tom Snyder

Contents

3 Connection Management

4 Transaction Management

5 Work Management

6 Using RAs for Inbound Communication

7 Managing Administered Objects

A OC4J Resource Adapter Configuration Files

B Third Party Licenses

Index

Preface

This document is an administrator's guide for Oracle Containers for J2EE (OC4J) customers using Oracle J2CA, the OC4J implementation of the standard J2EE Connector Architecture (J2CA). The standard governs the use of resource adapters for connecting to and accessing backend Enterprise Information Systems.

Oracle J2CA in Oracle Application Server 10*g* Release 3 (10.1.3.1.0) adheres to the *J2EE Connector Architecture Specification, Version 1.5*.

This preface contains the following sections:

Audience

Most information in this manual is for resource adapter configuration, so OC4J administrators are the primary audience. Secondary audiences include J2EE application component providers and resource adapter providers. See "Roles and Target Audiences" on page 1-13 for related information.

You should be familiar with the current version of the *J2EE Connector Architecture Specification*, produced by Sun Microsystems, although some of its concepts are reviewed here. There is a link to that specification near the end of this preface. Readers should also be familiar with concepts of Java Message Service (JMS) and message-driven beans (MDB).

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. Accessibility standards will continue to evolve over time, and Oracle 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 more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see the following Oracle resources.

Additional OC4J documents:

- *Oracle Containers for J2EE Developer's Guide*

  This document discusses items of general interest to developers writing an application to run on OC4J—issues that are not specific to a particular container such as the servlet, EJB, or JSP container. (An example is class loading.)

- *Oracle Containers for J2EE Deployment Guide*

  This document covers information and procedures for deploying an application to an OC4J environment. This includes discussion of the deployment plan editor that comes with Oracle Enterprise Manager 10*g*.

- *Oracle Containers for J2EE Configuration and Administration Guide*

  This document discusses how to configure and administer applications for OC4J, including use of the Oracle Enterprise Manager 10*g* Application Server Control Console, use of standards-compliant MBeans provided with OC4J, and, where appropriate, direct use of OC4J-specific XML configuration files.

- *Oracle Containers for J2EE Servlet Developer's Guide*

  This document provides information for servlet developers regarding use of servlets and the servlet container in OC4J, including basic servlet development and use of JDBC and EJBs.

- *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide*

  This document provides information about JavaServer Pages development and the JSP implementation and container in OC4J. This includes discussion of Oracle features such as the command-line translator and OC4J-specific configuration parameters.

- *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference*

  This document provides conceptual information as well as detailed syntax and usage information for tag libraries, JavaBeans, and other Java utilities provided with OC4J. There is also a summary of tag libraries from other Oracle product groups.

- *Oracle Containers for J2EE Services Guide*

  This document provides information about standards-based Java services supplied with OC4J, such as JTA, JNDI, JMS, JAAS, and the Oracle Application Server Java Object Cache.

- *Oracle Containers for J2EE Security Guide*

  This document (not to be confused with the *Oracle Application Server Security Guide*) describes security features and implementations particular to OC4J. This includes information about using JAAS, the Java Authentication and Authorization Service, as well as other Java security technologies.

- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*

  This document provides information about Enterprise JavaBeans development and the EJB implementation and container in OC4J.

Oracle Application Server Web services documents:

- *Oracle Application Server Web Services Developer's Guide*

  This document describes Web services development and configuration in OC4J and Oracle Application Server.

- *Oracle Application Server Advanced Web Services Developer's Guide*

  This document describes topics beyond basic Web service assembly. For example, it describes how to diagnose common interoperability problems, how to enable Web service management features (such as reliability, auditing, and logging), and how to use custom serialization of Java value types.

  This document also describes how to employ the Web Service Invocation Framework (WSIF), the Web Service Provider API, message attachments, and management features (reliability, logging, and auditing). It also describes alternative Web service strategies, such as using JMS as a transport mechanism.

- *Oracle Application Server Web Services Security Guide*

  This document describes Web services security and configuration in OC4J and Oracle Application Server.

Oracle TopLink documents:

- *Oracle TopLink Getting Started Guide*
- *Oracle TopLink Developer's Guide*

Additional Oracle Application Server documents:

- *Oracle Application Server Administrator's Guide*
- *Oracle Application Server Security Guide*
- *Oracle Application Server Performance Guide*

Oracle Enterprise Manager 10*g* Application Server Control online help topics, available through the Application Server Control Console:

- Task and conceptual topics
- Context-sensitive topics

Oracle JDeveloper documentation:

- Oracle JDeveloper online help
- Oracle JDeveloper documentation on the Oracle Technology Network:

http://www.oracle.com/technology/products/jdev/index.html

Resources from Sun Microsystems:

- Web site for the J2CA 1.5 specification:

  http://java.sun.com/j2ee/connector/download.html

- Web site for general J2EE 1.4 documentation:

  http://java.sun.com/j2ee/1.4/docs/

- Web site for J2EE 1.4 Javadoc:

  http://java.sun.com/j2ee/1.4/docs/api/index.html

Conventions

The following text conventions are used in this document.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

What to Know About Connector Architecture

A typical enterprise application must access data on some sort of backend information system. A problem encountered by businesses today, when they want to adapt their applications to Web-based technology, is that their data is on older, legacy systems that were not designed to be Web-accessible.

For enterprise application integration, application servers are a natural point of integration between Web-based applications and legacy systems. J2EE Connector Architecture (J2CA) offers a standard framework for accessing these legacy systems through J2EE-compliant application servers such as Oracle Containers for J2EE (OC4J). Oracle J2CA, the OC4J 10.1.3 implementation of J2CA, supports version 1.5 of the connector architecture.

This chapter offers an overview of key J2CA concepts and issues, including the following topics:

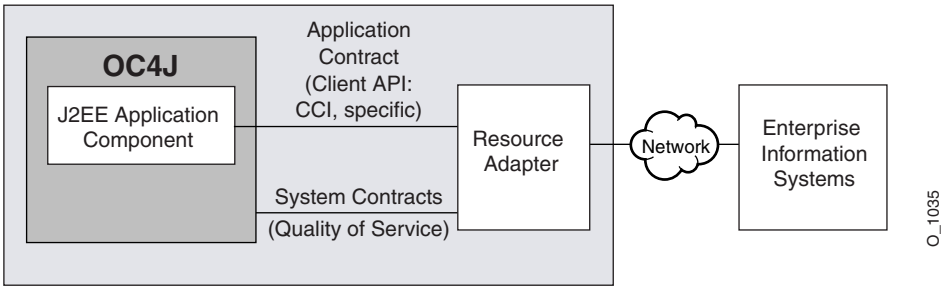
> **Note:** For further information about topics discussed here and elsewhere in this manual, refer to the *J2EE Connector Architecture Specification*. Version 1.5 is current as of the OC4J 10.1.3 implementation and is available at the following location:
>
> `http://java.sun.com/j2ee/1.4/docs/`

Introduction to the J2EE Connector Architecture

The J2EE Connector Architecture defines a standard architecture for bidirectional connectivity between the J2EE platform and heterogeneous legacy systems, referred to as *enterprise information systems*. This is accomplished through components known as *resource adapters*. These terms are more fully defined shortly. The basic architecture is shown in Figure 1–1.

The architecture supplies a mechanism for an enterprise information system vendor (or third party) to provide a standard resource adapter to be used by J2EE applications in accessing the information system, and for an application server vendor to provide the support that allows a standard resource adapter to be plugged in to the application server.

Figure 1–1 J2EE Connector Architecture



The rest of this section covers the following topics:

- What Is an Enterprise Information System?
- Connecting to an EIS: What Is a Resource Adapter?

What Is an Enterprise Information System?

An *enterprise information system*, or EIS, is a heterogeneous storage and retrieval system for enterprise data. Examples include enterprise resource planning (ERP) systems, customer relationship management (CRM) systems, mainframe transaction processing (TP) systems, other databases, and legacy applications that are not written in the Java programming language.

Legacy EISs have typically evolved over a significant amount of time through a significant amount of effort. Replacing these systems is usually not viable, so a mechanism is required to connect to them as is, preferably in a standard, reusable way.

Connecting to an EIS: What Is a Resource Adapter?

A *resource adapter* is a software driver that an application server or an application client uses to connect to a specific EIS. An example of a resource adapter is a JDBC driver to connect to a relational database. Similarly, an ERP system or CRM system might include a resource adapter.

Initial solutions to the need for resource adapters involved technologies that were proprietary and nonportable. Without the J2EE Connector Architecture, there would have to be a custom resource adapter to connect each type of application server to each type of EIS vendor.

With the J2EE Connector Architecture acting as a standard resource adapter framework, each application server requires just a single implementation supporting the connector architecture specification, and each EIS requires just a single resource adapter that can plug in to any compliant application server.

In addition, the initial proprietary solutions did not necessarily support services such as connection management and pooling, transaction management, and security. The J2EE Connector Architecture supports all these services and more, using standard mechanisms.

Speaking in a little more detail now, we define a resource adapter as the aggregate of classes and meta data necessary for communication between a particular EIS and a compliant J2EE application server. A resource adapter plugs into an application server, running within the application server address space.

New Resource Adapter Support Features in This OC4J Release

For those who are familiar with previous OC4J implementations of the connector architecture, this section summarizes enhancements in version 1.5 of the architecture and in the OC4J implementation.

New J2EE Connector Architecture Version 1.5 Features

Version 1.5 of the J2EE Connector Architecture represents a significant upgrade from version 1.0, including the addition of features for inbound communication. (Only outbound communication was supported in version 1.0.) Specifically, these new features, supported by the OC4J 10.1.3 implementation, include the following:

- Lifecycle management contract
- Work management contract
- Message inflow contract
- Transaction inflow contract
- Support for multiple implementation classes for managed connection factories

These and other contracts are summarized in "What Are the J2EE Connector Architecture System Contracts?" on page 1-3.

Additional New OC4J Resource Adapter Features

In addition to supporting the new J2EE Connector Architecture version 1.5 features, the OC4J 10.1.3 implementation provides the following enhancements:

- Connection pooling enhancements
- New connection-sharing features
- Support for updates to the OC4J-specific configuration files without stopping the server
- Closer integration with the JAAS security implementation for runtime permissions of resource adapters and for container-managed sign-on.
- Two-phase commit for resource adapters
- New metrics for connections and work management thread pools
- Deployment of multiple versions of the same resource adapter, with each version available as a shared library to all applications. An application can be configured to import a specific version of the resource adapter and load all classes from the resource adapter
- Deployment, undeployment, or reployment of a resource adapter without restarting the `default` application

What Are the J2EE Connector Architecture System Contracts?

In the J2EE Connector Architecture, J2EE components communicate with resource adapters using *system contracts* that specify standard support in several key areas of

functionality. Each of these contracts is partly implemented by an application server and partly implemented by a resource adapter so they can interact and collaborate in a standard way. (These contracts are also referred to as *quality of service* contracts.)

Here is a summary of the contracts:

- *Connection management* enables an application component to connect to an EIS and enables an application server to use connection pooling for these connections. (Do not confuse this with JDBC connection pooling, which is a separate mechanism.) See Chapter 3, "Connection Management" for details.
- *Transaction management* enables an application server to use a transaction manager to handle transactions across multiple EISs or *resource managers*. (A resource manager manages shared EIS resources). See Chapter 4, "Transaction Management" for details.
- *Security management* provides authentication, authorization, and secure communication between an application server and an EIS. Also see "What Are the Security Features of the J2EE Connector Architecture?" on page 1-6. For details about OC4J security for EIS connections, see the *Oracle Containers for J2EE Security Guide*.
- *Lifecycle management* allows an application server to manage the startup and shutdown of a resource adapter, including a mechanism for bootstrapping a resource adapter when it is deployed or when the application server starts up, and for notifying a resource adapter when it is undeployed or when the application server shuts down. Also see "Resource Adapter Lifecycle: Startup and Shutdown" on page 2-11 for related discussion.
- *Message inflow* allows a resource adapter to deliver messages to endpoints (such as message-driven beans) in an application server. Message delivery is asynchronous. This support does not rely on any particular message provider and has a wide range of uses, including with Java Message Service (JMS) or Java API for XML Messaging (JAXM), for example. Also see "Introduction to the Message Inflow Contract" on page 6-2.
- *Transaction inflow* allows an imported transaction to be propagated to an application server by a resource adapter. Also see "Introduction to Imported Transactions and the Transaction Inflow Contract" on page 6-3.
- *Work management* allows a resource adapter to perform tasks through the use of units of work submitted to an application server and executed by a work manager. Such tasks may include calling application components or monitoring network endpoints, for example. The application server uses separate threads to execute different work units, and the resource adapter is spared from having to manage the threads itself. See Chapter 5, "Work Management" for additional information.

What Are the Scenarios for Communication Through Resource Adapters?

In the J2EE Connector Architecture, communication between an application component and an EIS can originate from either party, and a resource adapter may support either or both of the following:

- *Outbound communication*—communication initiated by the application component
- *Inbound communication*—communication initiated by the EIS

This section compares outbound and inbound communication and briefly discusses messaging systems as they are relevant to inbound communication.

Outbound Versus Inbound Communication Through Resource Adapters

In outbound communication, the resource adapter simply allows an application to connect to the EIS and communicate with it through some API—perhaps using an EIS-specific API, or perhaps through the standard Common Client Interface (CCI) described later in this chapter. The resource adapter is a passive library in this case; communication is synchronous and is initiated by the application. An application component can use the API (EIS-specific or CCI) to synchronously execute an EIS function or retrieve the results of one. (The resource adapter itself might use some proprietary mechanism behind the scenes to make this possible.) The application server enforces the transactional semantics associated with the connection.

In inbound communication, the resource adapter allows an entity or event outside the application server to initiate activity. An EIS might accomplish this by writing a message to a queue, for example. Communication is asynchronous and is initiated by the EIS. A resource adapter supporting inbound communication will typically use the work management contract and message inflow contract. A work unit associated with the resource adapter coordinates receipt of incoming communication from the EIS, delivery of this communication to an appropriate recipient (message endpoint) within the J2EE container, and then delivery of a response with desired results to the EIS, as appropriate. The application server creates and manages threads that are assigned to the work units, assigns message endpoint instances to handle communication delivery, and enforces the transactional semantics associated with delivery.

Most EISs use a messaging system for initiating communication with applications that reside outside the EIS, such as an application running in OC4J. For this reason, inbound resource adapters are typically associated with a messaging system, although it is possible for an inbound adapter to use some proprietary mechanism such as a simple socket connection to listen for communication initiated by an EIS.

There is more information about messaging systems, and message endpoints, in the next section, "Message Provider Pluggability Using the J2EE Connector Architecture".

Figure 1–2 depicts the communication flow for outbound communication, also indicating where the relevant system contracts govern.

Figure 1–2 Outbound Communication

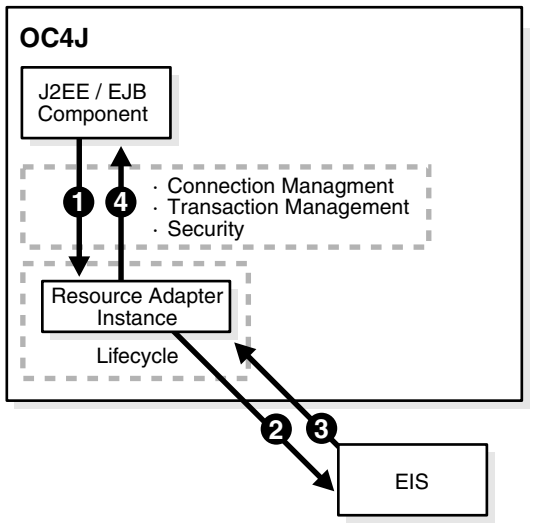
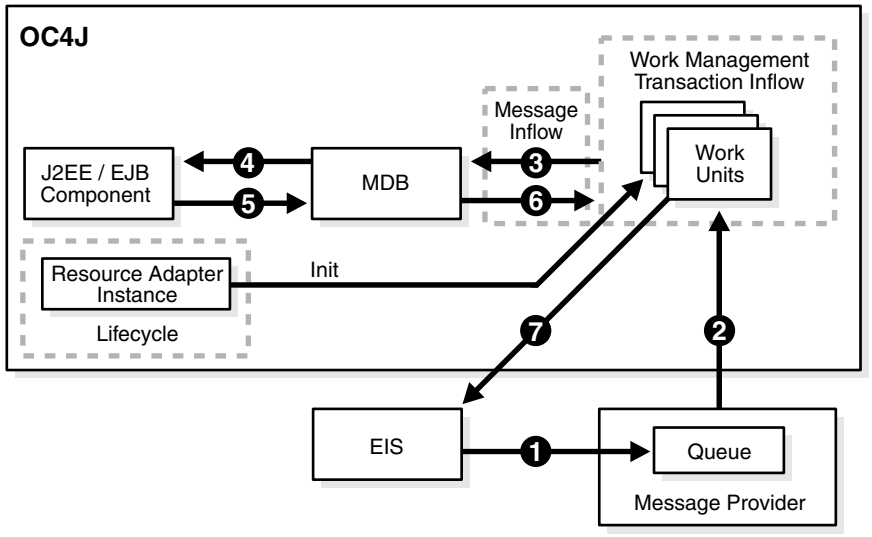


Figure 1–3 depicts the communication flow for inbound communication, also showing where the relevant system contracts govern.

Figure 1–3 Inbound Communication



Message Provider Pluggability Using the J2EE Connector Architecture

The J2EE 1.4 specification supports pluggability for a range of messaging systems, including JMS, through J2CA. A resource adapter intended for use with messaging typically supports one messaging system, or message provider, although there is no restriction on the number.

An outbound resource adapter provides the capability to send and receive messages in a synchronous manner using outbound communication. An inbound resource adapter provides the capability to asynchronously deliver messages from the message provider to a message endpoint, which is a message-driven bean (MDB) component in the application server. A bidirectional resource adapter provides both functions.

Note that most resource adapters that support inbound communication are based on a standard messaging system, such as a JMS implementation, so that the EIS can initiate communication in a standard manner.

For a brief introduction to a generic JMS resource adapter provided by Oracle, see "Introducing Oracle JMS Support and Generic JMS Resource Adapter" on page 1-12.

> **Note:** Resource adapters are now the recommended vehicle for plugging a messaging system into an application server.

What Are the Security Features of the J2EE Connector Architecture?

There are two separate aspects of security in a J2EE application that uses resource adapters:

- The J2CA security contract controls access to an EIS.
- *Security permissions* control access to application server resources, such as classloaders, threads, and sockets. These permissions appropriately restrict the behavior of an application running within OC4J.

The following sections summarize these topics:

- Summary of the Security Contract

Summary of the Security Contract

To ensure secure interactions between a J2EE application and an EIS, the J2EE Connector Architecture allows application components to associate a security context with connections established to the EIS. This is accomplished through the J2CA security contract, between an application server and a resource adapter. This contract extends the connection management contract with functionality relating to secure connections. It also works in conjunction with the standard Java Authentication and Authorization Service (JAAS), supporting standard JAAS interfaces.

The security contract includes features for the following:

- Propagating a security context, or subject, directly from a J2EE component to a resource adapter (for component-managed sign-on)
- Propagating a security context, or subject, from an application server to a resource adapter (for container-managed sign-on)

The security contract supports two particular authentication mechanisms:

- The commonly used "basic password" mechanism relies on a user name / password pair, contained together in a password credential object. The application server passes this object to the resource adapter for authentication.
- The Kerberos version 5 mechanism ("Kerbv5" for short) is an authentication protocol distributed by the Massachusetts Institute of Technology. This mechanism uses a "generic credential" object that encapsulates credential information such as a Kerberos ticket. The application server passes this object to the resource adapter for verification.

Sign-on from a J2EE application to an EIS can be managed either by the application component or by the J2EE container (OC4J). Component-managed sign-on must be set up programmatically and does not involve OC4J-specific configuration. Container-managed sign-on can be set up either declaratively, through OC4J-specific configuration without any programming requirements, or programmatically, involving a combination of OC4J-specific configuration and programming requirements. Programmatic container-managed sign-on can use either a "principal mapping class" or a JAAS login module.

See the *Oracle Containers for J2EE Security Guide* for details about all these features.

Security Permissions

An application server, such as OC4J, must provide a set of security permissions for use by a resource adapter executing in the managed runtime environment. The resource adapter must have appropriate and explicit permission to perform any sensitive operations, such as accessing resources that the application server controls (a classloader, for example).

If a resource adapter will require specific permissions, they must be indicated through `<security-permission>` elements in the `ra.xml` file. These permissions, however, can be overridden in an OC4J environment through `<security-permission>` elements in the `oc4j-connectors.xml` file. See "<security-permission>" on page A-23.

See version 1.5 of the *J2EE Connector Architecture Specification* for additional information.

What Are the Interface Libraries of the J2EE Connector Architecture?

The J2EE Connector Architecture includes the following interface libraries:

- Common Client Interface (CCI)

  This is a set of interfaces that can optionally be implemented by a resource adapter to provide a standard client-facing interface to use in accessing the associated EIS from J2EE components. If implemented, it is essentially a contract between a J2EE application and a resource adapter. Its use is not mandated by the connector architecture specification—a vendor-specific API can be implemented instead (such as a JDBC API for access to a database). However, to simplify access to a range of EISs from a range of J2EE components, support of the CCI by resource adapters is recommended. Such support guarantees that a resource adapter can be plugged into any J2EE-compliant development tool or enterprise application integration framework.

- Service Provider Interface (SPI)

  This is a set of interfaces, partly implemented by an application server and partly by a resource adapter, to provide the interface between the application server and resource adapter to manage physical connections to the EIS and to support the J2CA system contracts. The SPI interfaces are discussed throughout this book as appropriate, in the course of discussing the contracts.

The CCI and SPI interface libraries are in the packages `javax.resource.cci` and `javax.resource.spi`, respectively.

A resource adapter that implements the CCI provides a standard way for J2EE components to access the associated EIS. This simplifies the task of accessing multiple EISs from a single J2EE component.

The CCI can be used directly in a client application to access an EIS; however, it is more typical for a tools vendor to expose the CCI in the form of software developer kits (for runtime) or wizards (for design time) and for client applications to access an EIS through the tool. A resource adapter that supports the CCI can be plugged in to enterprise tools and other enterprise application integration frameworks in a standard way.

Communication through the CCI entails creating and executing an *interaction* to execute functions in the EIS. The sequence is to get a connection to the EIS, create an interaction, then execute the interaction. An interaction is represented by a class, provided with the resource adapter, that implements the CCI `Interaction` interface. This interface specifies an overloaded `execute()` method for performing operations on the EIS. An interaction object is obtained through an EIS connection and maintains an association with that connection. There are three kinds of interactions:

- `SYNC_SEND_RECEIVE`: Send a synchronous request to the EIS and receive an associated synchronous response. This is a standard request/response mechanism.
- `SYNC_SEND`: Send a synchronous request to the EIS, but without there being an associated synchronous response.
- `SYNC_RECEIVE`: Receive a synchronous response from the EIS, but without there having been an associated synchronous request. One scenario would be to use a `SYNC_SEND` interaction to start a long job, then use a `SYNC_RECEIVE` interaction sometime later, by which time the results should be ready.

The CCI `InteractionSpec` interface provides the mechanism for holding properties for interactions. (Also see "Example: What Is an Interaction Spec?" on page 7-2.)

Notes:

- You can find out whether a resource adapter supports CCI by examining the `<connection-interface>` element in the standard `ra.xml` deployment descriptor and seeing if it has a value of `javax.resource.cci.Connection`.
- This document does not go into detail about the CCI because its typical users, tools vendors, are not among our primary target audiences; however, there is additional information about interfaces related to connection management in "Summary of the Connection Management Contract" on page 3-14. For more information about the CCI and related `ra.xml` configuration, refer to the *J2EE Connector Architecture Specification*.

What Are the Packaging and Deployment Features?

The classes, interfaces, descriptors, and other resources that compose a resource adapter are packaged in a RAR file—a Java archive file with the `.rar` extension—for deployment. A RAR file must include at least the following:

- The `ra.xml` configuration file for the resource adapter
- One or more JAR files containing the resource adapter implementation

It can be deployed in either of two ways:

- Within an EAR file, in which case the resource adapter is available only to applications within the EAR file
- By itself, in which case the resource adapter is referred to as *standalone* and is associated with the OC4J default application and available to all applications in the server instance

  (OC4J is installed with a default configuration that includes a default application, also known as the global application. The `default` application is, by default, the parent of all other J2EE applications in OC4J, except Application Server Control Console.)

  Each standalone resource adapter is represented as a shared library, which, by default, is available to all applications. All code sources of a standalone resource adapter are added to a dedicated, shared loader that will be imported by all applications unless they are explicitly configured otherwise. When multiple versions of a standalone resource adapter are deployed, an application can be configured to import a specific resource adapter, so all resource adapter classes are loaded from the same adapter.

In addition to an `ra.xml` file, there must be an `oc4j-ra.xml` configuration file for each deployed resource adapter. This is an OC4J-specific file that you can include in the RAR file, or OC4J will create it automatically during deployment. Finally, the `oc4j-connectors.xml` file, also OC4J-specific, is an enumeration of all resource adapters associated with an application, and may contain some additional configuration as well. For resource adapters deployed in an EAR file, there is a single `oc4j-connectors.xml` file to enumerate the resource adapters, and OC4J will create it automatically if it is not found in the EAR file. For standalone resource adapters, there is a single `oc4j-connectors.xml` file in the OC4J configuration directory and associated with the OC4J default application to enumerate the standalone resource adapters. OC4J will also create this file if it does not already exist.

(See Appendix A, "OC4J Resource Adapter Configuration Files", for information about the OC4J-specific configuration files.)

Here are the contents of a sample RAR file:

```
META-INF/ra.xml
META-INF/oc4j-ra.xml
howto.html
images/icon.jpg
ra.jar
cci.jar
```

For this example, assume the following:

- The `ra.jar` file contains the resource adapter implementation.
- The resource adapter exposes the CCI APIs, contained in `cci.jar`, for its client interface.

Applications may need to access adapter-specific classes that are bundled in the RAR file. In the case of standalone resource adapters, these classes are available to all applications that are deployed within OC4J. In the case of resource adapters deployed in an EAR file, the classes are available only to modules deployed in the same EAR file.

Here are the contents of a sample EAR file (with only `oc4j-connectors.xml` and `myRar.rar` being related to resource adapters):

```
META-INF/application.xml
META-INF/oc4j-connectors.xml
myRar.rar
myWar.war
myEjb.jar
```

Deploying Resource Adapters

You can use any JSR-88-compliant tool to deploy resource adapters, including Application Server Control. (See "A Brief Overview of OC4J Administration" on page 2-1 for additional information.) Use the Application Server Control Console deployment plan editor to specify OC4J-specific parameter settings.

During deployment, OC4J decompresses the RAR file and takes the following actions regarding OC4J-specific deployment descriptor files:

- Creates a deployment directory for the resource adapter, of the following form:

  `j2ee/`*`instance`*`/application-deployments/`*`app_name`*`/`*`ra_name`*

  The OC4J instance, *`instance`*, is always `home` in a standalone environment and `home` by default in an Oracle Application Server environment. For a standalone resource adapter, the application name, *`app_name`*, is `default` and the resource adapter name, *`ra_name`*, is what you specify during deployment. For a resource adapter deployed in an EAR file, the application name is what you specify during deployment, and the resource adapter name is the RAR file name without the `.rar` extension.

  A RAR name must be unique within an OC4J instance. If you attempt to deploy a RAR that has the same file name as a RAR that was already deployed in an OC4J instance, the exception `NullPointerException` occurs.

- Generates an empty `oc4j-ra.xml` file if you did not provide one in the RAR file, placing it in the resource adapter deployment directory (*`ra_name`*).
- For a standalone resource adapter, adds a `<connector>` entry in the `oc4j-connectors.xml` file that is associated with the OC4J default application, creating the file if it does not already exist.
- For a resource adapter deployed in an EAR file, if you did not provide an `oc4j-connectors.xml` file in the EAR file, creates `oc4j-connectors.xml` in the application deployment directory (*`app_name`*) and adds a `<connector>` entry.

Each resource adapter must have a unique name when it is deployed:

Deploying, undeploying , or redeploying a resource adapter does not require a restart of the `default` application.

- The resource adapters within an application EAR file are deployed when the EAR file is deployed. If no name is specified in the `oc4j-connectors.xml` file, then the name is the same as the RAR archive name.
- A unique name must be provided for each standalone resource adapter during deployment.

See the *Oracle Containers for J2EE Deployment Guide* for further information.

Importing Standalone Resource Adapters

When an application is deployed, the application imports all standalone resource adapters that were previously deployed, by default.

When a standalone resource adapter is deployed, all running applications that were previously deployed, except the `default` application, are asked to import the resource adapter. So, an application that is dependent on a standalone resource adapter can be deployed before the resource adapter as long as the application does not attempt to use the resource adapter prior to its deployment. Take special care when you deploy a standalone resource adapter after a dependent application because the application might have already loaded classes. Unexpected exceptions will occur if importing the standalone resource adapter causes previously loaded classes to be subsequently loaded by a different loader.

A resource adapter can look up and use another resource adapter. Because each standalone resource adapter has it's own classloader, it is necessary for standalone resource adapters to import other deployed standalone resource adapters and shared libraries. By default, a standalone resource adapter will import all previously deployed standalone resource adapters and all shared libraries. A standalone resource adapter must be deployed before any dependent standalone resource adapters.

Deploying Multiple Versions of a Standalone Resource Adapter

You can deploy multiple standalone resource adapters that contain classes of the same name. Because all standalone resource adapters are available to all applications by default, any application that uses a standalone resource adapter for which multiple *versions* are deployed must explicitly specify which of the versions it will use. The application must use only one version of a resource adapter for which there are multiple versions.

An application specifies which standalone resource adapters it will use in the configuration file `orion-application.xml`. For example, two standalone resource adapters that contain some of the same classes are deployed with the names

adapterA and adapterB. The following connector elements are in *J2EE_HOME*/config/oc4j-connectors.xml:

```
<connector name="adapterA" path="adapterAFileName.rar" >
<connector name="adapterB" path="adapterBFileName.rar" >
```

To configure an application to use only adapterA, you would add the following elements to the application's orion-application.xml file:

```
<imported-shared-libraries>
<remove-inherited name="adapterB"/>
</imported-shared-libraries>
```

Because both standalone resource adapters are imported by default, it is not necessary to explicitly import adapterA.

An alternative configuration follows:

```
<imported-shared-libraries>
<import-shared-library name="adapterA">
      <remove-inherited name="*"/>
</imported-shared-libraries>
```

Note: While it is possible to version shared libraries using a version number, it is currently not possible to do this with standalone resource adapters. Each deployed resource adapter must have a unique name.

Deploying a resource adapter to the default application will prevent the resource adapter from using any standalone resource adapter that is deployed as a shared library. Resources deployed as shared libraries are not imported by the default application.

Resource Adapters Provided with Oracle Application Server

Resource adapters provided with Oracle Application Server include an Oracle generic JMS adapter and third-party adapters.

Introducing Oracle JMS Support and Generic JMS Resource Adapter

JMS specifies an enterprise messaging API that enables the use of portable, message-based applications for communication in a J2EE environment. There are a number of different JMS providers with a range of guarantees regarding reliability and quality of service. OC4J currently includes a proprietary resource provider mechanism for plugging in different JMS providers.

In Oracle Application Server 10*g* Release 3 (10.1.3.1.0), Oracle itself provides two JMS implementations. One, Oracle JMS (OJMS), is the JMS interface to the Oracle Database Streams Advanced Queueing (AQ) feature. The other, OracleAS JMS, is a native Java implementation that provides file-based persistence and is tightly integrated with OC4J.

Oracle also provides a J2CA 1.5-compliant JMS resource adapter that enables OC4J-managed applications to have a unified mechanism to access any JMS provider, regardless of whether their level of J2CA support is at version 1.5. This Oracle JMS resource adapter, referred to as the "generic JMS resource adapter", does not use any Oracle proprietary APIs. Supported JMS implementations include, for example, OJMS,

OracleAS JMS, and third-party products such as IBM WebSphere MQ JMS, Tibco Enterprise for JMS, and SonicMQ JMS.

The Oracle generic JMS resource adapter is the recommended path for JMS usage in the OC4J 10.1.3 implementation. It is based on the J2CA 1.5 and JMS 1.1 and 1.02b standards and includes minimal customization for OC4J, and none for individual JMS providers. It is intended to seamlessly integrate any standard JMS implementation. (Note that the generic JMS resource adapter cannot typically provide optimal access to a particular JMS provider, given that many JMS providers support custom extensions.) It also has a number of distinguishing features in the following areas:

- JNDI mapping
- MDB integration (including dynamic adjustment to changing message load)
- Global transaction support (including standards-based support for transaction recovery)
- True generic JMS connection pooling
- Deployment convenience (including order independence)
- Lazy resolution of JMS operations (including start order independence, tolerance of dynamic management such as starts and stops of JMS providers, and connection retries in case of provider failure)
- Performance
- JSR-77 statistics

Typically, the Oracle generic JMS resource adapter is used in situations where the EIS being connected to is a JMS resource provider. However, it can also be used in situations where an EIS uses JMS messaging as a means of notifying J2EE application components. In this case, the JMS resource adapter (along with a JMS resource provider) can be used instead of the inbound communication features (if any) of the EIS-specific resource adapter. This two-adapter solution, where the EIS-specific adapter is used for outbound communication and the Oracle generic JMS resource adapter is used for inbound communication, enables bidirectional communication between the EIS and J2EE application where it may otherwise not be possible.

Refer to the *Oracle Containers for J2EE Services Guide* for details about the Oracle JMS implementation and JMS resource adapter.

Use of Third-Party Resource Adapters

You can use J2CA 1.5-compliant or 1.0-compliant third-party resource adapters with the OC4J 10.1.3 implementation, to connect to a variety of backend systems, such as SAP, PeopleSoft, J.D. Edwards, and Siebel.

Oracle provides some third-party resource adapters as part of the full Oracle Application Server suite of products.

Roles and Target Audiences

This section discusses engineering and technician roles that are addressed by the J2CA specification, which of these roles represent target audiences for this book, and which topics of this book are of interest to each target audience.

J2EE Connector Architecture Roles

The J2CA specification is addressed to the following roles:

- Resource adapter provider

  This is an expert in the technology relating to a particular EIS.

- J2EE application server provider and container provider

  These roles are addressed separately in the connector architecture specification, but Oracle fills both functions with Oracle Application Server, which includes the OC4J containers and related services.

- J2EE application component provider

  This is for J2EE components that access one or more EISs. Ideally, the component provider is programming against convenient Java interfaces produced by a software development tools vendor, based on the CCI.

- Enterprise tools vendor

  Products from tools vendors may include data-mining and function-mining tools to analyze the scope and structure of EIS data; analysis and design tools for design based on EIS data and functions; code generation tools to produce Java classes to access EIS data and functions; and deployment tools.

- Application assembler

  The assembler packages application components into deployable entities.

- Application deployer

  The deployer loads a deployable entity into a target environment, such as an OC4J instance.

- System administrator

  The administrator manages and configures the environment of J2EE containers, resource adapters, and EISs.

Primary and Secondary Audiences and Topics of Interest

Of the roles introduced in the preceding section, the primary audience for this document is system administrators—individuals who will administer the OC4J environment, including resource adapter configuration. Application component providers and resource adapter providers are secondary audiences.

Topics in this book may be of interest to these audiences as indicated in Table 1–1.

Table 1–1 Audiences and Topics of Interest

Audience	Topics
System administrators	Chapter 2, "Overview: Administering Resource Adapters"
	Chapter 3, "Connection Management"
	Chapter 4, "Transaction Management"
	Chapter 7, "Managing Administered Objects"
	Appendix A, "OC4J Resource Adapter Configuration Files"
Application component providers	Chapter 3, "Connection Management"
	Chapter 4, "Transaction Management"
	Chapter 6, "Using RAs for Inbound Communication"

Table 1–1 (Cont.) Audiences and Topics of Interest

Audience	Topics
Resource adapter providers	Chapter 3, "Connection Management" Chapter 4, "Transaction Management" Chapter 6, "Using RAs for Inbound Communication" Chapter 7, "Managing Administered Objects"

2

Overview: Administering Resource Adapters

This chapter provides a top-level discussion of how to administer your resource adapters, covering the following topics:

A Brief Overview of OC4J Administration

Before discussing resource adapter administration, we summarize OC4J features in the following areas:

OC4J Deployment and Configuration Features

OC4J supports the following standards for deploying and managing applications in a J2EE environment:

- *Java Management Extensions (JMX) 1.2* specification allows standard interfaces to be created for managing resources, such as services and applications, in a J2EE environment. The OC4J implementation of JMX provides a user interface that you can use to completely manage an OC4J server and applications running within it.
- *Java 2 Platform, Enterprise Edition Management Specification* (JSR-77) allows objects known as *MBeans* (managed beans) to be created for runtime management of applications in a J2EE environment. In OC4J, you can directly access MBeans through the System MBean Browser in Oracle Enterprise Manager 10*g*, but many of their properties are exposed in a more user-friendly way through other features of Enterprise Manager. These interfaces are summarized in "Summary of Resource Adapter MBeans and Administration" on page 2-9.
- *Java 2 Enterprise Edition Deployment API Specification* (JSR-88) defines a standard API for configuring and deploying J2EE applications and modules into a J2EE-compatible environment. The OC4J implementation includes the ability to create or edit a *deployment plan* containing the OC4J-specific configuration data needed to deploy a component into OC4J.

The OC4J deployment plan editor and System MBean Browser are exposed through Oracle Enterprise Manager 10*g* Application Server Control, referred to as *Application Server Control*. The user interface for this is the *Application Server Control Console*. Additionally, for convenience, many parameters corresponding to MBeans properties, including all key properties relating to resource adapters, are exposed through other pages of the Application Server Control Console.

You should generally use the Application Server Control Console to configure and deploy your OC4J applications (using the deployment plan editor), as well as for any runtime administration or reconfiguration. Avoid direct manipulation of OC4J MBeans or of OC4J-specific XML configuration files where possible. The XML files are updated automatically by OC4J when you use the Application Server Control Console. There may be deployment situations, however, where an `oc4j-ra.xml` or `oc4j-connectors.xml` property is not exposed through the Application Server Control Console. In these situations, directly creating the XML file for deployment may be the only option.

For general information about OC4J deployment, configuration, and administration, refer to the *Oracle Containers for J2EE Deployment Guide* and the *Oracle Containers for J2EE Configuration and Administration Guide*. For more information about Application Server Control, you can also refer to the introduction to administration tools in the *Oracle Application Server Administrator's Guide*.

Notes:

- Due to the inherently ambiguous semantics involved in changing certain configuration settings while a resource adapter is in use, some updates through MBeans, including updates through the Application Server Control Console, do not take effect until the resource adapter is restarted. When you make such a change through the Application Server Control Console, OC4J updates the console and corresponding XML file immediately, even though the new value might not take effect until after restart.
- If you redeploy a resource adapter with active endpoints without stopping it first, OC4J throws a DeployerException exception due to the active endpoints. Stop the resource adapter before redeploying it.

OC4J and Oracle Application Server Administration Tools

In either an Oracle Application Server or standalone OC4J environment, you can deploy and configure your J2EE applications and resource adapters in OC4J through Application Server Control, introduced in "OC4J Deployment and Configuration Features" on page 2-1. This is generally the preferred way to manage your applications, and is therefore emphasized in this document. You can deploy an application through the Application Server Control Console "Deploy" feature in the Applications tab that is accessible from the OC4J Home page. Application Server Control Console pages for resource adapter configuration are discussed throughout this document.

In standalone OC4J, you also have the option of using the command-line OC4J `admin_client.jar` tool to deploy and bind your J2EE applications.

Alternatively, if you use the Oracle JDeveloper tool to develop your application, you can use it to deploy the application and any resource adapters as well.

Also, in some cases and particularly during development, it may be necessary to configure aspects of an OC4J application through direct manipulation of OC4J-specific XML files. For this reason, reference documentation for these files is included in the OC4J documentation set. Elements and attributes of the `oc4j-connectors.xml` and `oc4j-ra.xml` OC4J-specific resource adapter configuration files are documented in Appendix A, "OC4J Resource Adapter Configuration Files".

See the *Oracle Containers for J2EE Deployment Guide* and *Oracle Containers for J2EE Configuration and Administration Guide* for general information about using the Application Server Control Console or `admin_client.jar` tool to deploy and manage your applications. There is also extensive online help for the Application Server Control Console.

Summary of Application Server Control Pages for Resource Adapters

The Application Server Control Console provides a Web-based user interface for deploying, configuring, and monitoring applications, as well as managing the OC4J instance and the Web services used by your applications. It is installed, preconfigured, and started automatically when you install the OC4J software (either in a standalone or Oracle Application Server environment), and is bound to whichever port the OC4J instance is using. In an Oracle Application Server environment, the port is usually 7777. In a standalone environment, the port is typically 8888; use this port of the appropriate host to access the console:

`http://`*`hostname`*`:8888`

See the online Help provided with Application Server Control Console for detailed instructions on using this interface.

The console is organized into functional areas for applications, administration, performance, and Web services. You can manage resource adapters through the applications area.

Resource adapter configuration procedures in this document start from the Resource Adapter Home page.

The rest of this section covers the following topics:

- How to Get to the Resource Adapter Home Page
- Contents of the Resource Adapter Home Page
- Summary of Pages You Can Reach from the Resource Adapter Home Page

How to Get to the Resource Adapter Home Page

To get to the home page for a standalone resource adapter:

1. From the OC4J Home page, select the Applications tab.
2. View "Standalone Resource Adapters".
3. Select the resource adapter of interest.

To get to the home page for a resource adapter that was deployed with an application:

1. From the OC4J Home page, select the Applications tab.
2. View "Applications".
3. Select the desired application.

4. From the resulting Application Home page, under "Modules", select the resource adapter module of interest.

Contents of the Resource Adapter Home Page

Table 2–1 summarizes the resource adapter general properties shown in the home page, noting what XML entities they correspond to or are related to.

Table 2–1 General Properties Shown in the Resource Adapter Home Page

Application Server Control Property	Corresponding or Related XML Entity	Description
Name	`name` attribute of a `<connector>` element in `oc4j-connectors.xml`	The name of the resource adapter, as determined during deployment for a standalone resource adapter (such as through the Application Server Control Console deployment page), or according to the RAR file name, without the `.rar` extension, for a resource adapter deployed within an EAR file.
Status	n/a	Specifies whether the resource adapter is currently up (running). This is according to runtime polling by OC4J.
Path	`path` attribute of a `<connector>` element in `oc4j-connectors.xml`	The directory path from which the RAR file was deployed.
Description	`<description>` subelement of the applicable `<connector>` element in `ra.xml`	An optional description of the resource adapter.
Vendor Name	`<vendor-name>` subelement of the `<connector>` element in `ra.xml`	The vendor that supplies the resource adapter.
EIS Type	`<eis-type>` subelement of the `<connector>` element in `ra.xml`	The EIS for which this resource adapter is designed (for example, OracleAS JMS).
Adapter Version	`<version>` subelement of the `<connector>` element in `ra.xml`	Version number of the resource adapter, as specified by the vendor.
JCA Specification Version	`<resourceadapter-version>` subelement of the `<connector>` element in `ra.xml`	J2EE specification level supported by the resource adapter (either 1.0 or 1.5).
License Required	`<license-required>` subelement of the `<license>` element, which is a subelement of the `<connector>` element in `ra.xml`	A Boolean value specifying whether a license is required for this resource adapter. (There is optionally a `<description>` subelement of `<license>` to describe the licensing terms.)

Table 2–2 summarizes the resource adapter service contract properties shown in the home page, noting what XML entities they correspond to or are related to.

Table 2–2 Service Contract Properties Shown in the Resource Adapter Home Page

Application Server Control Property	Corresponding or Related XML Entity	Description
Communication with EIS	`<inbound-resourceadapter>` and `<outbound-resourceadapter>` subelements of the applicable `<resourceadapter>` element in `ra.xml` **Note:** These are J2CA 1.5 subelements.	Indicates "inbound" if only `<inbound-resourceadapter>` subelements are found, "outbound" if only `<outbound-resourceadapter>` subelements are found, or "bidirectional" if both are found. **Note:** For J2CA 1.0, only outbound communication is possible.
Connection Definitions	`<connection-definition>` subelement of the applicable `<outbound-resourceadapter>` element in `ra.xml` **Note:** These are J2CA 1.5 elements.	Indicates the number of connection definitions (the number of `<connection-definition>` elements) there are for this resource adapter. **Note:** In J2CA 1.0, there can be only one connection definition for each resource adapter.
Transaction Support	`<transaction-support>` subelement of the applicable `<outbound-resourceadapter>` element in `ra.xml` **Note:** In J2CA 1.0, `<transaction-support>` is a subelement of `<resourceadapter>`.	Indicates the level of transaction supported by this resource adapter: `NoTransaction`, `LocalTransaction`, or `XATransaction` (global transaction). See "Resource Adapter Configuration for Level of Transaction Support" on page 4-4.
Authentication Mechanisms	`<authentication-mechanism>` subelement of the applicable `<outbound-resourceadapter>` element in `ra.xml` **Note:** In J2CA 1.0, `<authentication-mechanism>` is a subelement of `<resourceadapter>`.	Indicates the type of authentication mechanism supported by this resource adapter, such as `BasicPassword`. (The typical use of a user name and password for sign-on.)
Reauthentication Support	`<reauthentication-support>` subelement of the applicable `<outbound-resourceadapter>` element in `ra.xml` **Note:** In J2CA 1.0, `<reauthentication-support>` is a subelement of `<resourceadapter>`.	A Boolean value specifying whether the resource adapter supports reauthentication of an existing managed connection instance.

Table 2–3 discusses the message listener types shown in the home page, noting the corresponding XML entities. The elements mentioned are among the definitions under a `<messageadapter>` subelement of an `<inbound-resourceadapter>` element in the `ra.xml` file.

Table 2–3 Message Listener Types Shown in the Resource Adapter Home Page

Application Server Control Property	Corresponding or Related XML Entity	Description
Message Listener Type (an entry for each message listener supported by the resource adapter)	`<messagelistener-type>` subelement of `<messagelistener>` in `ra.xml` (one `<messagelistener>` element for each supported message listener)	The Java type of a supported message listener (such as `javax.jms.MessageListener`). **Note**: Multiple message listener types can be supported by a single resource adapter.

Summary of Pages You Can Reach from the Resource Adapter Home Page

Starting from the Resource Adapter Home page, you can reach the following:

- Connection Factories tab: Edit, create, delete, or monitor a connection factory (referring to a particular JNDI location); or edit, create, delete, or monitor a shared connection pool.
  - Create Connection Factory page: Choose the connection factory interface, then specify the JNDI location, mode of connection pooling (if any), and configuration property settings for a new connection factory.
  - Edit Connection Factory page: Set configuration properties, connection pooling, security, and other options for a connection factory.
    * General tab: View configuration properties for connections from the factory. For a connection factory with private connection pooling, you can link from this tab to the Private Connection Pool page to edit parameters such as minimum and maximum number of connections in the pool.
    * Security tab: Set principal mapping entries for declarative container-managed sign-on to the EIS.
    * Options tab: Specify the desired path to a log file, or set the user and password to sign on to the EIS for transaction recovery.
  - Connection Factory Metrics page: Displays connection factory and connection pool metrics to monitor connections.
  - Create Shared Connection Pool page: Set connection pooling parameters (such as minimum and maximum number of connections in the pool) for a new connection pool.
  - Shared Connection Pool page: Modify connection pooling parameters for an existing connection pool.
- Administered Objects tab: Create, edit, or delete an administered object.
  - Create Administered Object page: Specify the JNDI location and set configuration properties for a new administered object.
  - Administered Object page: View configuration properties for an existing administered object.
- Administration tab: Set configuration properties for the resource adapter instance, or view the deployment descriptor (`ra.xml`) or proprietary deployment descriptor. Viewing the proprietary deployment descriptor displays the following:
  - The `oc4j-connectors.xml` `<connector>` element for the resource adapter
  - The `oc4j-ra.xml` file for the resource adapter

These pages, and what you can do from them, are discussed in more detail later in this document. You can also find more information in the context-sensitive topics "Resource Adapter Home Page", "Resource Adapter Connection Factories Page", "Resource Adapter Administered Objects Page", and "Resource Adapter Administration Page" in the Application Server Control online help.

General Resource Adapter Administration Features

This section highlights general configuration features for a resource adapter, covering the following topics:

- Setting Properties of the Resource Adapter JavaBean
- Configuring the Use of Resource Adapter Native Libraries

Setting Properties of the Resource Adapter JavaBean

A resource adapter instance is a JavaBean, which has configurable properties according to the setup in the `ra.xml` file. Consider the following example:

```
<connector ... >
   <resourceadapter>
      <resourceadapter-class>
         oracle.j2ee.ra.jms.generic.JMSResourceAdapter
      </resourceadapter-class>
      <config-property>
         <config-property-name>lookupMethod</config-property-name>
         <config-property-type>java.lang.String</config-property-type>
         <config-property-value>resourceProvider</config-property-value>
      </config-property>
      <config-property>
         <config-property-name>resourceProviderName</config-property-name>
         <config-property-type>java.lang.String</config-property-type>
         <config-property-value>MQSeries</config-property-value>
      </config-property>
      ...
   </resourceadapter>
</connector>
```

This indicates that `lookupMethod` and `resourceProviderName` are properties of the resource adapter JavaBean.

You can specify new values for resource adapter configuration properties, using the Application Server Control Console as follows:

1. From the appropriate Resource Adapter Home page, choose the Administration tab.
2. From the Administration tab, go to the Configuration Properties task.
3. The resulting Configuration Properties page lists configuration properties for the particular resource adapter. For each property, the assembled value from `ra.xml` is listed, and the current deployed value from `oc4j-connectors.xml` is listed. Given the preceding `ra.xml` example, `lookupMethod` and `resourceProviderName` would be listed.
4. Specify new deployed values as desired. For example, assume `resourceProviderName` is listed with an assembled value of `MQSeries`, as in

the preceding example, but you want to change it to `MQSeries2`. Simply specify this in the "Deployed Value" field.

5. Apply the changes.

For more information about pages summarized in the preceding steps, see the context-sensitive topics "Resource Adapter Administration Page" and "Resource Adapter Configuration Properties Page" in the Application Server Control online help.

> **Note:** While Application Server Control enables you to edit configuration properties, a resource adapter may not support dynamic updates. If you try to edit a configuration property in this circumstance, an exception is thrown.

Any new (deployed) value is reflected in a `<config-property>` element in the `oc4j-connectors.xml` file, with appropriate `name` and `value` attributes, as summarized in Table 2–4. Note that `<config-property>` elements have different uses. In this case, `<config-property>` is a subelement of the `<connector>` element for the resource adapter, and that is a subelement of the top-level `<oc4j-connectors>` element in `oc4j-connectors.xml`.

Table 2–4 Resource Adapter JavaBean Configuration Properties

Application Server Control Property	Corresponding XML Entity	Description
As applicable for the resource adapter, and as indicated by configuration property Name, Type, Assembled Value, and Deployed Value. (The Name, Type, and Assembled Value are according to `ra.xml`.)	`<config-property>` `name` and `value` attributes for Name and Deployed Value	These attributes correspond to the name and desired deployed value of a resource adapter JavaBean property.

Configuring the Use of Resource Adapter Native Libraries

You can deploy a native library with a resource adapter and instruct OC4J where to find it. A native library is typically a `.dll` file in a Windows environment or a `.so` file in a Solaris environment, for example.

Native library locations are not configurable through Application Server Control Console. Instead, use the `<native-library>` element of the `oc4j-connectors.xml` file to specify a relative location for the library, where the relative path is according to the path of the library in the deployment RAR file for the resource adapter.

For the following RAR file, the path would be "`./`".

```
META-INF/ra.xml
META-INF/oc4j-ra.xml
howto.html
images/icon.jpg
ra.jar
cci.jar
mylib.dll
```

The configuration would therefore be:

```
<native-library path="./mylib.dll" />
```

Or, for this RAR file, the path would be "`/mydir/`":

```
META-INF/ra.xml
META-INF/oc4j-ra.xml
mydir/mylib.dll
howto.html
images/icon.jpg
ra.jar
cci.jar
```

And the configuration would be:

```
<native-library path="mydir/mylib.dll" />
```

The `<native-library>` element is also discussed in "<native-library>" on page A-21.

Summary of Resource Adapter MBeans and Administration

Standards-compliant MBeans play a role in OC4J runtime configuration. The following sections provide an overview:

- General Overview of OC4J MBean Administration
- Summary of OC4J Resource Adapter MBeans

General Overview of OC4J MBean Administration

OC4J support for the JMX specification allows standard interfaces to be created for managing resources dynamically, including resources relating to resource adapters, in a J2EE environment. The OC4J implementation of JMX provides a JMX client, the System MBean Browser, that you can use to manage an OC4J instance through MBeans that are provided with OC4J.

An MBean is a Java object that represents a JMX manageable resource. Each manageable resource within OC4J, such as an application or a resource adapter, is managed through an instance of the appropriate MBean. Each MBean provided with OC4J exposes a management interface that is accessible through the System MBean Browser in the Application Server Control Console. You can set MBean attributes, execute operations to call methods on an MBean, subscribe to notifications of errors or specific events, and display execution statistics.

> **Note:** This information is provided for reference, but key resource adapter configuration settings are exposed in a more user-friendly manner through other features of the Application Server Control Console. Procedures for particular tasks are described throughout this document.

To access the browser from the OC4J Home page, select the Administration tab and then, under the list of tasks, go to the JMX task "System MBean Browser". From the browser, you can do the following:

- Select the MBean of interest in the left-hand frame.

- Use the Attributes tab in the right-hand frame to view or change attributes. A settable attribute has a field where you can type in a new value. Then apply the change.
- Use the Operations tab in the right-hand frame to invoke methods on the MBean. Select the operation of interest. In the Operation window, you can it with specified parameter settings.
- Use the Notifications tab (where applicable) in the right-hand frame to subscribe to notifications. You can select each item for which you want notification, and then apply the changes.
- Use the Statistics tab (where applicable) in the right-hand frame to display execution statistics.

Be aware that MBeans and their attributes vary regarding when changes take effect. In the *runtime model*, changes take effect immediately. In the *configuration model*, some changes take effect when the resource is restarted, others when the application is restarted, and still others when OC4J is restarted. There is also variation in whether changes are persisted.

See the *Oracle Containers for J2EE Configuration and Administration Guide* for details. The System MBean Browser itself also provides information about the MBeans.

Summary of OC4J Resource Adapter MBeans

OC4J exports a set of MBeans for each resource adapter, to support administration during application runtime. Some OC4J MBeans are required in order to support the J2EE management specification, but may offer extended features. Other OC4J MBeans are Oracle extensions to the model.

Table 2–5 summarizes the OC4J implementations of MBeans, relating to resource adapters, that are required of an application server according to JSR-77. These implementations are in the `oracle.oc4j.admin.management.mbeans` package.

Notes:

- MBeans marked with a double-asterisk ("**") in the following tables have properties whose changes will not take effect until the resource adapter or OC4J (usually the resource adapter) is restarted.
- MBeans are self-documenting in the System MBean Browser, providing some documentation of MBean attributes, operations, and notifications (as applicable).

Table 2–5 Mandatory System MBeans for Resource Adapters

MBean	Description
ResourceAdapterModule **	Identifies a resource adapter archive (RAR file) as a whole, and allows configuration of properties across the archive (possibly extending across multiple resource adapters).
ResourceAdapter **	Identifies a single resource adapter across an archive.
JCAResource	Identifies a single configured resource from a resource adapter that is available to all components deployed within OC4J.

Table 2–5 (Cont.) Mandatory System MBeans for Resource Adapters

MBean	Description
JCAConnectionFactory **	Identifies a configured connection factory instance within a resource.
JCAManagedConnectionFactory	Identifies the managed connection factory associated with a connection factory instance.

Table 2–6 summarizes OC4J MBeans, relating to resource adapters, that are Oracle extensions. These MBean implementations are also in the `oracle.oc4j.admin.management.mbeans` package.

Table 2–6 Additional System MBeans for Resource Adapters

MBean	Description
WorkManagerResource	Identifies the work manager associated with a resource.
JCAEndpointActivation	Identifies an endpoint that has been activated to use this resource.
JCAMessageEndpointFactory	Identifies the endpoint factory that is associated with an endpoint activation.
JCAAdminObject	Identifies an administered object supported by this resource adapter.
JCAAdminObjectInstance **	Identifies a configured instance of one of the administered objects supported by this resource adapter.
JCASharedConnectionPool	Identifies one of the shared connection pools that is defined for this resource adapter.
JCAConnectionPool	Identifies one of the private connection pools that is defined for this resource adapter.
JCAConnectionDefinition	Identifies one of the connection definitions that is packaged with this resource adapter.

Resource Adapter Lifecycle: Startup and Shutdown

When a resource adapter is deployed, or when an application server, such as OC4J, is started, the application server requires a mechanism to "bootstrap" (load and initialize) an instance of the resource adapter. When a resource adapter is undeployed or otherwise stopped, or when an application server is stopped, the application server requires a mechanism to notify the resource adapter to stop functioning prior to being unloaded. These functional areas are covered by the J2CA lifecycle management contract.

This section introduces key aspects of this contract, and also covers how to manually stop a resource adapter. The following topics are covered:

- Key APIs of the Lifecycle Management Model
- Understanding the Resource Adapter Startup Process
- Understanding the Resource Adapter Shutdown Process
- Manually Shutting Down or Restarting a Resource Adapter in OC4J

Key APIs of the Lifecycle Management Model

A resource adapter is implemented by the resource adapter provider as a JavaBean class implementing the following SPI interface:

- `javax.resource.spi.ResourceAdapter`

The resource adapter provider deploys the JavaBean class as part of the overall resource adapter deployment, and also specifies configuration for `ResourceAdapter` instances in the standard `ra.xml` deployment descriptor.

The `ResourceAdapter` interface specifies the following methods that are called by the application server for resource adapter startup and shutdown, respectively:

- `void start(javax.resource.spi.BootstrapContext ctx)`
- `void stop()`

The `BootstrapContext` interface is implemented by the application server, with methods that give access to certain application server resources.

Understanding the Resource Adapter Startup Process

Resource adapter startup is initiated by OC4J and occurs in the following circumstances:

- When the resource adapter is deployed to OC4J
- When OC4J is restarted after the resource adapter has been deployed

OC4J takes the following steps for resource adapter startup:

1. Instantiates the `ResourceAdapter` JavaBean for this resource adapter, configuring its properties according to specifications in the `ra.xml` file.
2. Creates a `BootstrapContext` instance to pass to the `ResourceAdapter` instance.
3. Calls the `ResourceAdapter` JavaBean `start()` method, passing in the `BootstrapContext` instance.
4. Initializes and binds any connection factories. (See "Introduction to How EIS Connections Are Obtained" on page 3-1 for a definition of "connection factory".)
5. Initializes and binds any administered objects. (See "What Is an Administered Object?" on page 7-1 for a definition of "administered object".)

The `start()` method is the vehicle by which the resource adapter can execute initialization procedures, establish communications with the EIS, and possibly start the EIS service. Initialization procedures may include, for example, creating threads, setting up network endpoints, or creating objects (such as administered objects) specific to the resource adapter.

The `BootstrapContext` instance has methods giving access to application server resources, such as by returning a `java.util.Timer` instance to schedule future tasks, a `javax.resource.spi.work.WorkManager` instance to manage work units, or a `javax.resource.spi.XATerminator` instance for transaction completion or recovery. The `start()` method may use the work manager to submit work units for execution, or the XA terminator to complete imported transactions. (For information about work management and transaction inflow, see "Overview of the Work Management Contract" on page 5-1 and "Overview of Related Contracts for Inbound Communication" on page 6-2.)

Notes:

- The preceding discussion assumes a J2CA 1.5 environment. Only the step involving connection factories is relevant in a J2CA 1.0 environment.
- The `<start-order>` element in the `oc4j-connectors.xml` file determines the order in which resource adapters are loaded and started by OC4J within an application, or the order in which standalone resource adapters are loaded and started, as applicable. See "<start-order>" on page A-23.
- If the `start()` method throws an exception, OC4J considers the startup process to have failed and makes no further attempt to call the `start()` method on the same `ResourceAdapter` instance. Instead, it creates a new `ResourceAdapter` instance when it tries to start the resource adapter again.
- The OC4J `ResourceAdapter` MBean—which includes `start()`, `stop()`, and `restart()` methods to start, stop, or restart a resource adapter—uses the start and stop functionality of the resource adapter JavaBean. (See "Summary of Resource Adapter MBeans and Administration" on page 2-9 for information about OC4J MBeans.)

Understanding the Resource Adapter Shutdown Process

Resource adapter shutdown is initiated by OC4J and occurs in the following circumstances:

- When the resource adapter is undeployed
- When OC4J is shut down
- When you manually stop the resource adapter (described in the next section, "Manually Shutting Down or Restarting a Resource Adapter in OC4J")

Resource adapter shutdown is a two-phase process:

1. In the first phase, OC4J must ensure that all applications and operations that depend on the resource adapter are terminated before the resource adapter is shut down. This includes the following:
   - Deactivate any message endpoints that receive messages through this resource adapter.
   - Stop any applications that use this resource adapter.

   The first phase guarantees that application threads no longer use the resource adapter instance and that all application activities, including transactional activities, are completed.
2. In the second phase, begun only after completion of the first phase, OC4J calls the `ResourceAdapter` JavaBean `stop()` method to notify the resource adapter to stop functioning. This allows it to be safely unloaded.

   When the `stop()` method is called, the resource adapter performs any necessary cleanup and then performs an orderly shutdown. This may include the following:
   - Close network endpoints.
   - Relinquish threads.

- Release active work units.
- Allow any transactions in the commit phase to complete.
- Flush any cached data to the EIS.
- Unbind and clean up connection factories.
- Unbind and clean up any administered objects.

Notes:

- The preceding discussion assumes a J2CA 1.5 environment. In a J2CA 1.0 environment, all that is relevant is the unbinding and cleaning up of connection factories.
- After the shutdown steps, any application still holding handles to cached connection factories or administered objects may see unexpected behavior.
- When stopping a resource adapter, OC4J does not always stop dependent applications. Stop any applications that use a resource adapter before you stop it, to make sure all application activity completes.
- If you redeploy a resource adapter with active endpoints without stopping it first, OC4J throws a DeployerException exception due to the active endpoints. Stop the resource adapter before redeploying it.
- OC4J treats the resource adapter as nonfunctional after the `stop()` method is called. OC4J ignores exceptions from the `stop()` method
- To restart a resource adapter after it has been stopped, OC4J creates a new `ResourceAdapter` instance, then calls the `start()` method on that instance.

Manually Shutting Down or Restarting a Resource Adapter in OC4J

You can manually shut down a resource adapter in OC4J. In the Application Server Control Console, take the following steps from the appropriate Resource Adapter Home page:

1. Choose the Stop function.
2. In the resulting Confirmation page, say Yes to confirm the action.

After shutting down a resource adapter, you can restart it as follows, also from the Resource Adapter Home page:

1. Choose the Start function. (No further steps are necessary.) Status of the resource adapter is displayed on the home page.

3

Connection Management

This chapter provides details of how OC4J administrators set up the configuration for resource adapter connections, then summarizes the J2CA connection management contract and discusses how an application component obtains a connection. This includes the following topics:

> **Note:** See Summary of Application Server Control Pages for Resource Adapters on page 2-3 for an overview of the Application Server Control Console pages discussed here. Discussion in this chapter starts from the point where you have reached the appropriate Resource Adapter Home page.

Introduction to How EIS Connections Are Obtained

To use the resources of an EIS, a J2EE application component obtains a connection object, uses the underlying connection to conduct its business (reading or writing data as desired), then closes the connection.

Connection objects are obtained through a *connection factory*, which is implemented by the resource adapter provider. A connection factory object is registered in the JNDI namespace, through configuration steps that you take, and has a method that returns a connection object. The application component performs the JNDI lookup to retrieve the factory, then makes a request through the factory to obtain a connection. The factory delegates the request to OC4J. (See the *Oracle Containers for J2EE Services Guide* for information about the OC4J JNDI implementation.)

For a resource adapter that implements the CCI, the factory will be an instance of a class that implements the CCI (`javax.resource.cci`) `ConnectionFactory` interface, which specifies a method `getConnection()` that returns an instance of a class that implements the CCI `Connection` interface.

In the J2EE Connector Architecture, the functionality discussed here is specified in the connection management contract, which is more fully summarized in "Summary of the Connection Management Contract" on page 3-14. This contract specifies how to create connections to an EIS and how to set up JNDI configuration for connection factories. It also provides a general outline for how a J2EE container such as OC4J can support connection pooling, which is critical for efficient use of EIS resources and scalability of applications, and it specifies how to find matching physical connections in a connection pool. See "Configuring Connection Pooling in OC4J" on page 3-4 for information about how the administrator configures the OC4J implementation of connection pooling.

The next section, "Binding and Configuring a Connection Factory: Basic Settings", describes how the administrator sets up connection factories.

Binding and Configuring a Connection Factory: Basic Settings

Before an application component can use connections to an EIS, one or more connection factories must be configured. This section shows the most basic steps in configuring a resource adapter and binding it to JNDI. Further steps, such as configuring connection pooling for a connection factory, are discussed later in this chapter. Additional steps, such as configuring security, are discussed later in this manual.

"Obtaining Resource Adapter Connections" on page 3-14 shows how to use a connection factory in application code.

Create and Bind a Connection Factory

When you deploy a resource adapter, OC4J generates an `oc4j-ra.xml` file (unless one is packaged in the RAR file), using corresponding entries in the packaged `ra.xml` file as defaults.

For each connection type defined, OC4J generates a `<connector-factory>` element in `oc4j-ra.xml` during deployment. (In version 1.5 of the *J2EE Connector Architecture Specification*, a connection type corresponds to a `<connection-definition>` element in the `ra.xml` file.) Furthermore, when you configure a resource adapter, OC4J generates subelements under the `<connector-factory>` element for any settings you specify through the Application Server Control Console that supplement or override entries in `ra.xml`.

The unique identifier for a connection type in the `ra.xml` file is the connection factory interface. One interface is specified as part of each `ra.xml` connection definition. In `oc4j-ra.xml`, however, there can ultimately be multiple connection factories (that is, multiple `<connector-factory>` elements) using the same connection factory interface and therefore corresponding to the same `ra.xml` connection definition. The point of this would be to configure each connection factory with different property settings, as desired. For example, you can use different connection factories to connect to different servers. (Note that the JNDI location you specify for each connection factory must be unique to that connection factory.)

A key step in creating a connection factory is to *bind* it, where binding consists of specifying a JNDI location. The simplest scenario for this is to use the default *assembled values* for the configuration properties, meaning values that come from the `ra.xml` file. (We will assume for this discussion that they are specified in `ra.xml`, which is optional.) However, when you create a connection factory, you can specify new *deployment values* for the configuration properties.

You also specify whether you want connection pooling for the connection factory and, if so, whether to use a private pool or shared pool.

In the Application Server Control Console, use the following steps.

1. In the Connection Factories tab accessed from the Resource Adapter Home page for the appropriate resource adapter (the generic JMS resource adapter, for example, if it is deployed), choose the "Create" feature to start the process of binding a new connection factory.
2. In the Create Connection Factory: Select Interface page, choose the desired connection factory interface. For a J2CA 1.5 adapter, each choice corresponds to one and only one connection definition in the `ra.xml` file. For a version 1.0 adapter, there will be only one choice. (The generic JMS resource adapter, for example, has the choices `QueueConnectionFactory`, `XAQueueConnectionFactory`, `TopicConnectionFactory`, and `XATopicConnectionFactory`.)
3. In the resulting Create Connection Factory page, you can do the following:
   - Specify a JNDI location (mandatory).
   - Specify whether you want the connection factory to use no connection pool (the default), a private connection pool, or a shared connection pool. To use shared pooling, at least one shared pool must already exist, and you must choose which shared pool to use. See "Configuring Connection Pooling in OC4J" on page 3-4 and "Sharing Connection Pools" on page 3-10 for information about connection pools.
   - Optionally edit configuration properties of the connection factory.
4. Still in the Create Connection Factory page, choose the "Finish" feature to bind the connection factory.

As a result of this configuration, OC4J generates a `<connector-factory>` element in the `oc4j-ra.xml` file. Table 3–1 shows the properties that indicate the connection factory interface and JNDI location.

Also see the context-sensitive topic "Create Connection Factory: Select Interface Page" in the Application Server Control online help.

Table 3–1 Connection Factory Basic Properties

Application Server Control Property	Corresponding XML Entity	Description
Connection Factory Interface	`<connectionfactory-interface>` subelement	Java interface to use for the connection factory.
JNDI Location	`location` attribute	A JNDI location to which the connection factory object will be bound.
As applicable for the resource adapter, and as indicated by configuration property Name and Value attributes. Or, when you are editing, as indicated by Name, Assembled Value, and Deployed Value. (Name and Assembled Value are according to `ra.xml`.)	`<config-property>` `Name` and `Value` attributes	These attributes correspond to the name and desired deployed value of a connection factory configuration property.

Note: Binding a connection factory takes effect immediately; no restart of OC4J is necessary. (Note, however, that the binding of a connection factory that is preconfigured in an `oc4j-ra.xml` file that is packaged with the application occurs when the associated resource adapter is initialized, which follows resource adapter deployment or OC4J startup.)

Edit the Configuration Properties of an Existing Connection Factory

You can edit the settings of a connection factory that you previously created and bound.

In the Application Server Control Console, do the following:

1. In the Connection Factories tab accessed from the Resource Adapter Home page for the appropriate resource adapter, choose the JNDI location of a previously configured connection factory. Note that the JNDI location serves as the identifier of the connection factory and cannot be edited. The assembled values are from `ra.xml` entries.
2. In the General tab accessed from the resulting Edit Connection Factory page, you can specify new deployed values of any editable configuration properties. You can also configure the connection pool from here, if the connection factory was created with private or shared connection pooling enabled. See the next section, "Configuring Connection Pooling in OC4J", for information.
3. In the same page, apply the changes.

When you edit a connection factory, OC4J adds or updates `<config-property>` entries in `oc4j-ra.xml` to override previous settings in `ra.xml` or `oc4j-ra.xml`, as applicable.

Also see the context-sensitive topic "Resource Adapter Connection Factories Page" in the Application Server Control online help.

Note: You must restart OC4J for these changes to take effect.

Configuring Connection Pooling in OC4J

For performance and scalability, the J2EE Connector Architecture supports the implementation of connection pooling by any application server provider. This section discusses how to configure connection pooling in OC4J, beginning with basic configuration and then covering connection pooling enhancements added in the OC4J 10.1.3 implementation.

You determine whether to enable connection pooling when you create a connection factory, and you can configure the applicable pool when you edit the connection factory.

Notes:

- A connection pool is initialized during OC4J startup or when the resource adapter is deployed.
- A connection pool can be private to a single connection factory or shareable between multiple connection factories, but the configuration information discussed here generally applies in either case. See "Sharing Connection Pools" on page 3-10 for information about shareable pools.

The presence of a `<connection-pooling>` subelement under a `<connector-factory>` element in the `oc4j-ra.xml` file implies that connection pooling will be used for the corresponding connector factory (unless the `<connection-pooling>` element has the attribute setting `use="none"`). When you configure connection pooling through Application Server Control, OC4J generates the `<connection-pooling>` element along with a `<property>` subelement for each connection pooling property that you set.

If you do not configure connection pooling, OC4J exhibits the default behavior of creating a new physical connection whenever an application requests a connection. You can also explicitly specify no connection pooling for any given connector factory, through the Application Server Control Console.

Note: Because the J2EE Connector Architecture is general, rather than specific to databases, the connector architecture connection pooling interface differs significantly from the JDBC connection pooling interface.

The rest of the discussion for configuring connection pools is organized as follows:

- Enable Connection Pooling for a Connection Factory
- Configure a Connection Pool
- Pooling Scheme, Minimum and Maximum Connections, Initial Capacity
- Checking for Expired or Invalid Connections
- OC4J Support for Runtime Configuration of Connection Pools

Enable Connection Pooling for a Connection Factory

As noted in "Create and Bind a Connection Factory" on page 3-2, you choose whether to use connection pooling for a connection factory when you create the connection factory. During creation, you must specify whether you want the connection factory to use no connection pool (the default), a private connection pool, or a shared connection pool. For you to use shared pooling, at least one shared pool must already exist, and you must choose which shared pool to use. (See "Sharing Connection Pools" on page 3-10 for information about shared pools.)

Table 3–2 documents relevant properties in the Application Server Control Console Create Connection Factory page.

Table 3–2 Properties for Enabling or Disabling Connection Pooling

Application Server Control Property	Corresponding XML Entity	Description
No Connection Pool	`<connection-pooling>` with `use="none"`	Disables connection pooling for the connection factory you are configuring.
Use Private Connection Pool	`<connection-pooling>` with `use="private"`	Specifies the use of a private connection pool for the connection factory you are configuring.
Use Shared Connection Pool	`<connection-pooling>` with `use="shared"`	Specifies the use of a shared connection pool for the connection factory you are configuring.
(Indicated name of shared connection pool)	`<use-connection-pool>`	Names the shared connection pool to use.

Note: If an OC4J 9.0.4 version of `oc4j-ra.xml` is deployed to an OC4J 10.1.3 implementation server, any `<connection-pooling>` element receives an attribute setting of `use="private"` by default. This attribute was not in use in the OC4J 9.0.4 implementation. Other supported settings are "`none`", for no connection pooling, or "`shared`" for shared pools, as discussed later in this chapter.

Configure a Connection Pool

You can change the configuration of a connection pool by editing the connection factory that uses it, regardless of whether the pool is private or shared.

Use the Application Server Control Console, as follows:

1. From the Connection Factories tab, accessed from the Resource Adapter Home page, choose the appropriate JNDI location to select the connection factory of interest.
2. From the General tab of the resulting Edit Connection Factory page, select the shared connection pool or select "Private", as applicable.
3. In the resulting Private Connection Pool page (or Shared Connection Pool page for a shared pool), specify desired settings for the connection pool parameters. For information about these parameters, see the next section, "Pooling Scheme, Minimum and Maximum Connections, Initial Capacity", and see "Checking for Expired or Invalid Connections" on page 3-8.
4. Apply your changes.

Also see the context-sensitive topics "Private Connection Pool Page" and "Shared Connection Pool Page" in the Application Server Control online help.

Pooling Scheme, Minimum and Maximum Connections, Initial Capacity

Table 3–3 summarizes basic connection pooling settings supported since the OC4J 9.0.4 implementation. You can view or edit these in the Application Server Control Console Private Connection Pool page (or the Shared Connection Pool page, for shared pools). OC4J reflects each setting in a `<property>` subelement of the applicable `<connection-pooling>` element in `oc4j-ra.xml` (or the applicable `<connection-pool>` element for shared pools).

Table 3–3 Basic Connection Pool Properties

Application Server Control Property	Corresponding XML Entity	Description
Connection Pooling Scheme	`<property>` with `name="scheme"` and `value="dynamic","fixed",` or `"fixed_wait"`	How OC4J handles connection requests after the maximum number of connections is reached. This attribute supports a setting of "Dynamic", "Fixed", or "Fixed Wait", as described later in this table.
Maximum Connections	`<property>` with `name="maxConnections"` and `value` as appropriate	Desired maximum number of connections to maintain within a pool during program execution. If no value or a value of 0 is specified, then there is no limit on the number of connections.
Minimum Connections	`<property>` with `name="minConnections"` and `value` as appropriate	Desired minimum number of connections to maintain within a pool during program execution. The default value is 0.
Initial Capacity	`<property>` with `name="initial-capacity"` and `value` as appropriate	Desired number of connections for OC4J to create during initialization of the connection pool. If the specified initial capacity is less than the specified minimum number of connections, then any additional connections, once created, will not be removed from the pool until the number of connections exceeds the minimum. **Note:** Initial capacity applies to private connection pools only, not to shared connection pools. **Note:** It is possible that OC4J will be unable to open the specified number of connections due to lack of necessary information at initialization time, such as JNDI context.
Fixed Wait Timeout	`<property>` with `name="waitTimeout"` and `value` as appropriate	Maximum number of seconds to wait for an available connection if the maximum number of connections is exceeded and the Fixed Wait scheme is in effect. (Otherwise, this property is ignored.) The default is no timeout.

Here are descriptions of the three supported connection pooling schemes:

- "Dynamic" to always create a new connection and return it to the application, even if this violates the maximum limit. The next time a connection is closed, it is destroyed instead of being returned to the pool if the pool is still over the maximum limit. This is the default setting.
- "Fixed" to raise an exception when the application requests a connection after the maximum limit has been reached.
- "Fixed Wait" to block the application until a connection becomes available and is returned to the pool. If the "Fixed Wait Timeout" property is specified, then OC4J throws an exception if no connection becomes available within the specified wait timeout period.

The Initial Capacity parameter is useful if you anticipate that the demand for connections will be particularly high at startup. For example, if you have a connection

pool with Minimum Connections set to 5 and Maximum Connections set to 50, with an anticipated high demand at startup, you can set Initial Capacity to 25. Once the initial demand has passed, you can use the Inactive Connection Timeout parameter (described in the next section, "Checking for Expired or Invalid Connections") to allow the pool to be scaled back toward the minimum number of connections until the next peak usage period.

Note: For any `oc4j-ra.xml` file, if there is a `<connection-pooling>` element with `use="private"` but no connection pool properties specified, OC4J uses a default pool configuration with the Dynamic connection pooling scheme and no preset minimum or maximum number of connections. Also, no connections are pre-created when the connection pool is initialized.

Checking for Expired or Invalid Connections

The OC4J 10.1.3 implementation adds support for removing connections that have expired or been marked invalid in a connection pool. By checking unused connections in a pool, OC4J can ensure that connections are valid when passed to J2EE applications.

Connections can become invalid due to a network timeout or other internal error. To enable OC4J to check for invalid connections, the resource adapter provider must implement the optional `ValidatingManagedConnectionFactory` interface to specify which connections are invalid. This interface, in the `ManagedConnectionFactory` implementation class underlying the connection factory, is part of the of the SPI (`javax.resource.spi`) package. See "Summary of the Connection Management Contract" on page 3-14 for information about managed connection factories and how they are used.)

You can enable the checking features as follows:

- How often OC4J will check for invalid or expired connections is according to the "Expired Connection Cleanup" parameter.
- The timeout value for a connection to expire is according to the "Inactive Connection Timeout" parameter. After a connection in the pool is unused for that amount of time, it will be discarded.

These parameters, from the Application Server Control Console Private Connection Pool page (or Shared Connection Pool page, for a shared pool) are summarized in Table 3–4.

Depending on how you set these parameters, OC4J will do no checking, check only for invalid connections (if connections are set to never expire), or check for both invalid and expired connections (according to the specified timeout).

Each setting is reflected by OC4J in a `<property>` subelement of the applicable `<connection-pooling>` element.

Table 3–4 Settings to Check for Invalid or Expired Connections

Application Server Control Property	Corresponding XML Entity	Description
Expired Connection Cleanup	`<property>` with `name="inactivity-timeout-check"` and `value="never"`, `"periodic"`, `"piggyback"`, or `"all"`	When to check for expired or inactive connections. Supported values are `"never"`, `"periodic"`, `"piggyback"` (when a new connection is fetched), and `"all"` (periodically and when a new connection is fetched), as described in the following text. Set the `inactivity-timeout-check` property in the `oc4j-ra.xml` file before deploying the resource adapter.
Inactive Connection Timeout	`<property>` with `name="inactivity-timeout"` and `value` as appropriate	The desired connection timeout, in seconds, as a positive integer, or 0 for connections to never expire. Negative values are disallowed.

Use "Expired Connection Cleanup" as follows:

- `"never"` to never check for expired or invalid connections. This effectively disables checking for both expired connections and invalid connections. If you specify a large number for maximum connections to accommodate peak usage, you can use `"never"` to trim the size of the connection pool to the in-use set of connections. If inactivity almost never occurs in your connection pool because the number of connections constantly in use is close to the number of maximum connections, specifying `"never"` can help optimize server performance.
- `"periodic"`, the default value, to check after a set amount of time, repeatedly, such as every 10 minutes. The period used for checking is according to the `granularity` setting of the OC4J Task Manager (an interval in milliseconds). This is configurable through the JMX Management Console in Application Server Control and reflected in the value of the `taskmanager-granularity` attribute of the `<application-server>` element in the OC4J `server.xml` file. Be aware that any changes to this parameter will affect other OC4J tasks and OC4J performance. See *Oracle Containers for J2EE Configuration and Administration Guide* for information about Task Manager granularity.
- `"piggyback"` (when a new connection is fetched) to check only whenever a new connection is requested.
- `"all"` (periodically and when a new connection is fetched) to combine periodic checking with checking whenever a connection is requested.

When a connection is removed from a pool, the destroy() method is called.

> **Note:** If removal of invalid or expired connections brings the number of connections in the pool below the desired minimum, OC4J will create new connections as appropriate.

OC4J Support for Runtime Configuration of Connection Pools

You can make changes to the parameters discussed in the preceding sections during program execution (with the exception of initial capacity, which takes effect only at OC4J startup). The changes are dynamic; they do not require a restart of OC4J. Here is a list of parameters for which runtime changes are supported, with notes about related OC4J behavior:

- For "Connection Pooling Scheme", the new scheme applies with the next connection request by an application. If the original scheme was "Fixed Wait", the new scheme will also take effect for any pending connection requests; and if the new scheme is "Fixed", this results in an exception for any connection request that was waiting for an available connection. If the original scheme was "Dynamic", and the number of connections is greater than the desired maximum at the time you change the scheme to "Fixed" or "Fixed Wait", then OC4J attempts to observe the maximum by closing and removing any unused connections from the pool.
- For "Maximum Connections", if the new value is less than the existing number of connections in the pool, OC4J will close any unused connections. If the number of in-use connections is still greater than the new maximum value, then as connections are closed by the application, they will not be returned to the pool until the number of connections is below the new maximum.
- For "Minimum Connections", if the new value is greater than the existing number of connections in the pool, OC4J will create enough new connections to satisfy the new minimum.
- For "Wait Timeout", the new value applies with the next connection request, assuming the scheme is "Fixed Wait".
- For "Inactive Connection Timeout", the new value applies the next time OC4J checks for expired connections. If the value was 0 and is now a positive integer, OC4J will start checking for expired connections. In this case, if the value of "Expired Connection Cleanup" is `"never"`, it will be changed to `"periodic"`.
- For "Expired Connection Cleanup", if the old value was `"periodic"` or `"all"` (periodically and when a new connection is fetched), the next periodic check may still be carried out.

Sharing Connection Pools

Beginning with the OC4J 10.1.3 implementation, there is support for multiple connection factories to share a single connection pool if all the factories are for the same resource adapter and use the same managed connection factory implementation class. This allows users to better manage the number of concurrent connections to the same type of EIS, or to utilize the same connection pool for all connection factories used with a single resource adapter.

This section documents shared connection pools, covering the following topics:

- Create and Configure a Shared Connection Pool
- Edit and Reconfigure a Shared Connection Pool
- Use a Shared Connection Pool

Create and Configure a Shared Connection Pool

To create a shareable connection pool, take the following steps in the Application Server Control Console:

1. Go to the Connection Factories tab, accessible from the Resource Adapter Home page.
2. Under "Shared Connection Pools", choose the "Create" function.
3. In the resulting Create Shared Connection Pool page, specify a desired name for the pool, such as `SharedPool1`. This is how it will be referred to later, when you want to use it for a connection factory. This property is summarized in Table 3–5.
4. Set parameters as desired. They are used in the same ways as for a private connection pool, as described in "Pooling Scheme, Minimum and Maximum Connections, Initial Capacity" on page 3-6 and "Checking for Expired or Invalid Connections" on page 3-8.
5. Apply your changes.

For a shared connection pool, OC4J will generate a `<connection-pool>` element directly under the root `<oc4j-connector-factories>` element in the `oc4j-ra.xml` file, instead of generating a `<connection-pooling>` element under a `<connector-factory>` element, as for a private pool associated with just one connection factory.

The supported properties for a shared pool are the same as for a pool that is specific to a particular connection factory. In the `oc4j-ra.xml` file, however, these would appear in `<property>` subelements of a `<connection-pool>` element, instead of in `<property>` subelements of a `<connection-pooling>` element (which is a subelement of the `<connector-factory>` element).

Also see the context-sensitive topic "Create Shared Connection Pool Page" in the Application Server Control online help.

Table 3–5 Shared Connection Pool Name Property

Application Server Control Property	Corresponding XML Entity	Description
Connection Pool Name (applicable only for a shared connection pool)	`<connection-pool>` `name` attribute	Desired name of a connection pool, to make it available for sharing between connection factories.

> **Note:** Runtime changes are supported for shared connection pools, as they are for private connection pools. See "OC4J Support for Runtime Configuration of Connection Pools" on page 3-10.

Edit and Reconfigure a Shared Connection Pool

To edit an existing shared connection pool:

1. You can get to the Shared Connection Pool page in the Application Server Control Console in either of two ways:
   - From the Connection Factories tab, accessible from the Resource Adapter Home page, select the pool you want to edit from the list under "Shared Connection Pools".
   - From the Connection Factories tab, select a connection factory that uses the shared connection pool you want to edit.
2. In the resulting Edit Connection Factory page, select the shared connection pool.
3. In the resulting Shared Connection Pool page, set parameters as desired.

Parameters are used in the same ways as for a private connection pool, as described in "Pooling Scheme, Minimum and Maximum Connections, Initial Capacity" on page 3-6 and "Checking for Expired or Invalid Connections" on page 3-8.

4. Apply your changes.

Also refer to the topic "Shared Connection Pool Page" in the Application Server Control online help.

Note: You cannot change the name of an existing shared connection pool.

Use a Shared Connection Pool

As noted in "Enable Connection Pooling for a Connection Factory" on page 3-5, you specify whether you want to use shared connection pooling (or private connection pooling or no connection pooling) when you first create a connection factory, in the Create Connection Factory page of the Application Server Control Console. If you choose to use shared connection pooling, you must also specify which shared pool to use.

When a connection factory is to use a shared connection pool, OC4J updates the `oc4j-ra.xml` file, adding a `<connection-pooling>` element for the appropriate connection factory (under the appropriate `<connector-factory>` element) with an attribute setting of `use="shared"` and a `<use-connection-pool>` subelement with a value that specifies the name of the shared pool. This name corresponds to the `name` attribute of the `<connection-pool>` element that specifies the shareable pool being used.

Notes:

- If a `<connection-pooling>` element has a `<use-connection-pool>` subelement, OC4J ignores any `<property>` subelements that would relate to a private pool.
- If a `<connection-pooling>` element has an attribute setting of `use="shared"` but there is no `<use-connection-pool>` subelement, OC4J will throw an exception.
- After specifying that a connection factory use a shared connection pool, you must restart OC4J for the change to take effect.

Configuring OC4J Logging for Connection Factories

The J2EE Connector Architecture includes optional features for system-level error logging and tracing for a particular managed connection factory. Using these features, an application server can detect error conditions in a resource adapter or its EIS and use error information in debugging problems. The application server manages the association between a log writer and a managed connection factory.

In OC4J, through Application Server Control, you can enable logging for a resource adapter by specifying a log file for a corresponding connection factory. Accomplish this through the following steps in the Application Server Control Console:

1. Go to the Connection Factories tab from the appropriate Resource Adapter Home page.

2. Choose the connection factory for which you want to enable logging.
3. From the resulting Edit Connection Factory page, go to the Options tab.
4. Specify the desired absolute or relative path and file name for a log file.
5. Apply your changes.

This setting is reflected in the `<log>` subelement of a `<connector-factory>` element in the `oc4j-ra.xml` file, as summarized in Table 3–6.

Also see the context-sensitive topic "Edit Connection Factory Options Page" in the Application Server Control online help.

Table 3–6 Log File Properties

Application Server Control Property	Corresponding XML Entity	Description
Log File	`path` attribute of the `<file>` subelement of the `<log>` element	This specifies the absolute or relative path and name of a log file where OC4J will write logging and tracing messages relating to the resource adapter and its EIS.

A relative path is relative to the resource adapter deployment directory. As noted in "What Are the Packaging and Deployment Features?" on page 1-9, this is as follows, where *`instance`* is the name of the OC4J instance (`home` by default in an Oracle Application Server environment and always `home` in a standalone environment); *`app_name`* is the deployed application name (or `default` for a standalone resource adapter); and *`ra_name`* is the deployed resource adapter name (as specified during deployment for a standalone resource adapter, or according to the RAR file name, without the `.rar` extension, for a resource adapter deployed within an EAR file):

`j2ee/`*`instance`*`/application-deployments/`*`app_name`*`/`*`ra_name`*

Specifying either "`mylog.log`" or "`./mylog.log`", for example, results in the following log file location:

`j2ee/instance/application-deployments/`*`app_name`*`/`*`ra_name`*`/mylog.log`

Or "`mydir/mylog.log`" results in the following:

`j2ee/instance/application-deployments/`*`app_name`*`/`*`ra_name`*`/mydir/mylog.log`

Notes:

- If the specified directory for the log file does not exist, or if OC4J does not have permission to write to the specified directory, then OC4J does not enable logging and outputs a warning message.
- If the specified directory exists but the file does not exist, OC4J creates the file and enables logging.
- Messages written by OC4J to the log include those for error conditions that occur during deployment of the resource adapter or when OC4J attempts to start the resource adapter.
- The log file location is also passed on to managed connections.

Obtaining Resource Adapter Connections

This section shows a JNDI lookup of the connection factory that was configured in "Create and Bind a Connection Factory" on page 3-2.

This example also depends on the associated JNDI configuration in the `ejb-jar.xml` and `orion-ejb-jar.xml` files. Following is a corresponding sample `ejb-jar.xml` entry. This file would be packaged in the application EAR file.

```
<resource-ref>
   <res-ref-name>eis/myEIS</res-ref-name>
   <res-type>javax.resource.cci.ConnectionFactory</res-type>
   <res-auth>Application</res-auth>
</resource-ref>
```

Use the Application Server Control Console to specify a resource reference mapping entry to link the JNDI location previously bound, `eis/ConnectionFactory`, to the JNDI name in the `ejb-jar.xml` file, `eis/myEIS`. This generates an entry in `orion-ejb-jar.xml` such as the following:

```
<resource-ref-mapping name="eis/myEIS" location="eis/ConnectionFactory" />
```

See the *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* for information about EJB configuration for resource references.

Here is the sample code to look up the connection factory:

```
try
{
 Context ic = new InitialContext();
 ConnectionFactory cf = (ConnectionFactory) ic.lookup("java:comp/env/eis/myEIS");
} catch (NamingException ex) {
 ex.printStackTrace();
}
```

For a connection factory that implements the CCI `ConnectionFactory` interface, use the `getConnection()` method to create a connection to the EIS. This returns a connection object that implements the CCI `Connection` interface.

```
Connection conn = cf.getConnection();
```

Summary of the Connection Management Contract

"Introduction to How EIS Connections Are Obtained" on page 3-1 provides a brief overview of how a connection is obtained. Now that we have covered the necessary configuration steps earlier in this chapter for connection factories and connection pooling, this section goes into more detail about the specifications of the connection management contract for obtaining a connection. We outline the steps as follows:

1. The application component performs a JNDI lookup to obtain a connection factory object. The connection factory is implemented by the resource adapter provider. If the resource adapter implements the CCI, this is through a class that implements the CCI `ConnectionFactory` interface. The connection factory is registered in JNDI by the OC4J administrator.
2. The application component calls a method of the connection factory to request a connection to the EIS. If the resource adapter implements the CCI, this is through a `getConnection()` call to a `ConnectionFactory` object. Additionally, in the CCI, the application can optionally pass a *connection spec* object in the `getConnection()` call, to pass in properties specific to the connection request. A connection spec is an instance of a class, typically a JavaBean, that implements the

CCI `ConnectionSpec` interface. Connection spec properties must be defined through the getter and setter method pattern.

3. The connection factory delegates the connection request to OC4J, which maintains a connection pool. Specifically, the connection factory calls the `allocateConnection()` method of the OC4J connection manager, an instance of a class that implements the SPI `ConnectionManager` interface.
4. The OC4J connection manager obtains a connection, using the resource adapter *managed connection factory* as appropriate, which implements the SPI `ManagedConnectionFactory` interface. A physical connection to the EIS is referred to as a *managed connection*, represented by an instance of a class that implements the SPI `ManagedConnection` interface. OC4J attempts to obtain a connection as follows:
   a. First, a new logical connection can be created from a physical connection already in use if it is being used within the same transaction scope and is not marked as unshareable (by a `<res-sharing-scope>` subelement of a `<resource-ref>` element in the standard `ejb-jar.xml` or `web.xml` file).
   b. If the first scenario is not feasible, OC4J looks for an available managed connection in the connection pool. The OC4J connection manager calls the `matchManagedConnection()` method of the managed connection factory. If a suitable `ManagedConnection` object is found in the pool, then OC4J will use it to satisfy the connection request.
   c. If neither of the first two scenarios is feasible, the OC4J connection manager calls the `createManagedConnection()` method of the managed connection factory. This method creates a new physical connection and returns a new `ManagedConnection` object to OC4J to represent the physical connection. OC4J places the new managed connection in the pool and will use it to satisfy the connection request.
5. OC4J obtains an application-level connection handle from the managed connection. The handle is an instance of a class implemented by the resource adapter. If the resource adapter implements the CCI interface, then this class implements the CCI `Connection` interface.
6. OC4J returns the connection handle to the application.
7. The application component uses the connection handle to access the EIS (then closes the connection when through).

Additionally, OC4J registers a connection event listener—an instance of a class that implements the SPI `ConnectionEventListener` interface—with the `ManagedConnection` instance for the physical connection. OC4J accomplishes this by calling the `addConnectionEventListener()` method of the managed connection. The purpose of this is to allow OC4J to be notified of events related to the managed connection, which is useful in managing the connection pool.

Metrics for Resource Adapter Connections

OC4J provides performance metrics for resource adapter connection factories and connection pools, based on the Oracle Dynamic Monitoring Service (DMS) as well as on industry standard metrics specified in the *Java 2 Platform, Enterprise Edition Management Specification* (JSR-77). The following sections describe these metrics and how to access them:

- Viewing Resource Adapter Connection Pool Metrics

- Descriptions of Connection Pool Configuration Metrics
- Descriptions of Connection Factory Performance Metrics
- Descriptions of Connection Pool Performance Metrics
- Troubleshooting with Resource Adapter Connection Pool Metrics

(For more information about OC4J performance metrics, see the topic "Summary of the OC4J Performance Metrics" in the Application Server Control online help.)

Note: DMS adds performance-monitoring features to a number of Oracle Application Server components, including OC4J. The goal of DMS is to provide information about runtime behavior through built-in performance measurements so that users can diagnose, analyze, and debug any performance problems. DMS provides this information in a package that can be used at any time, including during live deployment. Data are published through HTTP and can be viewed with a browser. For general information about how to use DMS, the built-in DMS metrics that are available, and other OC4J performance considerations, refer to the *Oracle Application Server Performance Guide*.

Viewing Resource Adapter Connection Pool Metrics

You can view metrics for resource adapter connection factories and pools by taking the following steps in the Application Server Control Console:

1. From the appropriate Resource Adapter Home page, go to the Connection Factories tab.
2. For the appropriate JNDI location, corresponding to the connection factory of interest, choose the "Monitor" feature.
3. The resulting Connection Factory Metrics page displays statistics such as wait time, time of use, connection acquisition rate, and connection release rate.

Metrics listed under "General" are for the particular connection factory that you chose, corresponding to a single JNDI location. Metrics listed under "Connection Pool" are for the connection pool as a whole, which can be either private, where it is used by a single connection factory corresponding to a single JNDI location, or shared, where it is used by multiple connection factories corresponding to multiple JNDI locations.

The following metrics are available under the "General" category. See "Descriptions of Connection Factory Performance Metrics" on page 3-17 for information.

- Connection Wait Time (seconds)
- Connection Use Time (seconds)
- Connection Acquisition Rate (per second)
- Connection Release Rate (per second)

The following metrics are available under the "Connection Pool" category. See "Descriptions of Connection Pool Performance Metrics" on page 3-18 for more information.

- Pool Type (private or shared)
- Pool Name
- Connection Wait Time (seconds)

- Connection Use Time (seconds)
- Connection Creation Rate (per second)
- Connection Closure Rate (per second)
- Connections in Use
- Available Connections
- Waiting Threads
- Connection Error Rate (per second)

Also see the context-sensitive topic "Connection Factory Metrics Page" in the Application Server Control online help.

Descriptions of Connection Pool Configuration Metrics

As listed in Table 3–7, there are DMS metrics that correspond to connection pool configuration parameters that are described in "Configuring Connection Pooling in OC4J" on page 3-4. XML entities in the table refer to subelements of the applicable `<connection-pooling>` element in the `oc4j-ra.xml` file.

Table 3–7 OC4J Connection Pool Configuration Metrics

DMS Metric Name (and Sensor Type)	Application Server Control Property	Corresponding XML Entity
maxPoolSize (State)	Maximum Connections	`<property>` with `name="maxConnections"`
minPoolSize (State)	Minimum Connections	`<property>` with `name="minConnections"`
scheme (State)	Connection Pooling Scheme	`<property>` with `name="scheme"`
waitTimeout (State)	Fixed Wait Timeout	`<property>` with `name="waitTimeout"`
inactivityTimeout (State)	Inactive Connection Timeout	`<property>` with `name="inactivity-timeout"`
inactivityTimeoutCheck (State)	Expired Connection Cleanup	`<property>` with `name="inactivity-timeout-check"`

Descriptions of Connection Factory Performance Metrics

Table 3–8 lists and describes OC4J DMS metrics for a connection factory and notes the corresponding metrics in the Application Server Control Connection Factory Metrics page (under "General") and in the J2EE Management Specification, as applicable. A connection factory corresponds to a single JNDI location.

These statistics are of interest regardless of whether connection pooling is used.

Table 3–8 OC4J Connection Factory Performance Metrics

DMS Metric Name (and Sensor Type)	Application Server Control Metric	J2EE Management Specification / JSR-77 Statistics (and Type)	Description
waitTime (PhaseEvent)	Connection Wait Time	JCAConnectionStats.getWaitTime() (TimeStatistics)	Average number of seconds spent waiting for a connection from this connection factory.
useTime (PhaseEvent)	Connection Use Time	JCAConnectionStats.getUseTime() (TimeStatistics)	Average number of seconds spent using a connection.
createCount (Event)	Connection Acquisition Rate	JCAConnectionStats.getCreateCount() (CountStatistics)	Number of connection handles acquired (expressed as per second in Application Server Control).
closeCount (Event)	Connection Release Rate	JCAConnectionStats.getCloseCount() (CountStatistics)	Number of connection handles released (expressed as per second in Application Server Control).

Note: When you do not use a connection pool, acquiring a connection is equivalent to creating a connection, and releasing a connection is equivalent to closing a connection. When you do use a connection pool, a single connection handle may be acquired and released multiple times between creation and closure.

Descriptions of Connection Pool Performance Metrics

Table 3–9 lists and describes OC4J DMS metrics for a connection pool, and notes the corresponding metrics in the Application Server Control Connection Factory Metrics page (under "Connection Pool") and in the J2EE Management Specification, as applicable. A connection pool may be private, in which case it corresponds to a single connection factory and JNDI location, or it may be shared, in which case it corresponds to multiple connection factories and JNDI locations.

These statistics are of interest only when connection pooling is used.

Table 3–9 OC4J Connection Pool Performance Metrics

DMS Metric Name (and Sensor Type)	Application Server Control Metric	J2EE Management Specification / JSR-77 Statistics (and Type)	Description
poolName (State)	Pool Name	n/a	Name of the connection pool.
waitTime (PhaseEvent)	Connection Wait Time	JCAConnectionPoolStats.getWaitTime() (TimeStatistics)	Average number of seconds spent waiting for a connection from this connection pool. For a shared pool, this is for all the connection factories in the pool.
useTime (PhaseEvent)	Connection Use Time	JCAConnectionPoolStats.getUseTime() (TimeStatistics)	Average number of seconds spent using a connection. For a shared pool, this is for all the connection factories in the pool.
createCount (Event)	Connection Creation Rate	JCAConnectionPoolStats.getCreateCount() (CountStatistics)	Number of connection handles created (expressed as per second in Application Server Control). For a shared pool, this is for all the connection factories in the pool.
closeCount (Event)	Connection Closure Rate	JCAConnectionPoolStats.getCloseCount() (CountStatistics)	Number of connection handles closed (expressed as per second in Application Server Control). For a shared pool, this is for all the connection factories in the pool.
freePoolSize (State)	Available Connections	JCAConnectionPoolStats.getFreePoolSize() (BoundedRangeStatistics)	Number of available connections in the pool.
poolSize (State)	None, but equivalent to Connections in Use + Available Connections	JCAConnectionPoolStats.getPoolSize() (BoundedRangeStatistics)	Size of connection pool—number of free connections plus number of connections in use.
waitingThreadCount (PhaseEvent)	Waiting Threads	JCAConnectionPoolStats.getWaitingThreadCount( ) (BoundedRangeStatistics)	Number of threads waiting for a connection.
expiredCount (Event)	n/a	n/a	Number of expired connections removed from the pool.

Table 3–9 (Cont.) OC4J Connection Pool Performance Metrics

DMS Metric Name (and Sensor Type)	Application Server Control Metric	J2EE Management Specification / JSR-77 Statistics (and Type)	Description
invalidCount (Event)	n/a	n/a	Number of connections determined by the resource adapter to be invalid.
requestTimeoutCount (Event)	n/a	n/a	Number of connection requests that failed due to timeout.
errorCount (Event)	Connection Error Rate	n/a	Number of connection error events (expressed as per second in Application Server Control).

Troubleshooting with Resource Adapter Connection Pool Metrics

OC4J connection metrics discussed in the preceding sections are for your use in learning the tendencies of your resource adapters and application server environment. There are no specific desired value ranges for these metrics. Over time, you will learn what ranges are normal for your environment, and what ranges may indicate trouble.

Typically, you should pay particular attention to wait times, the number of waiting threads, and the connection error rate.

Note: An additional tool for troubleshooting connections is the OC4J system property `jca.connection.debug`, which you can use to output diagnostic information for J2CA connections. See the *Oracle Containers for J2EE Configuration and Administration Guide* for general information about OC4J system properties.

4

Transaction Management

This chapter covers key aspects of transactions and the J2CA transaction management contract for outbound connections. This contract leverages the standard J2EE programming model for transaction management, allowing access by an application to multiple disparate EISs through standard interfaces in the context of a single, coordinated transaction.

The following topics are covered here:

- Introduction to Transactions and Transaction Management
- Overview of Key Interfaces Used in Transaction Management
- Transaction Support in the OC4J Connector Implementation
- OC4J Transaction Setup and Cleanup
- Understanding Connection Sharing in OC4J
- Understanding and Configuring Transaction Recovery
- Understanding J2CA Connection Wrapping

For additional information about OC4J transaction support, see the Java Transaction API chapter of the *Oracle Containers for J2EE Services Guide*.

Introduction to Transactions and Transaction Management

The topic of transactions concerns how updates and manipulations of data are managed in order to ensure the integrity of the data. A transaction may involve just a single EIS or multiple EISs. Particularly in the latter case, strict rules must be followed to coordinate the updates and ensure that data is not left in an inconsistent state in the event of problems during the transaction. For example, if a transaction involves transferring money from one account to another, you want to avoid a situation where the balance is updated in only one account.

The following subsections provide a quick introduction or refresher for transactions and transaction management:

- Transaction Characteristics and Scopes
- Relationship with Enterprise JavaBeans Technology
- Relationship with Java Transaction API Technology
- Local Transaction Management
- Global Transaction Management
- Resource Adapter Configuration for Level of Transaction Support

Transaction Characteristics and Scopes

A *transaction* is a unit of work, typically consisting of multiple operations, between an application component and one or more EISs to update or manipulate data. Transactions must have four basic characteristics: *atomicity* (ensuring that either all operations of a transaction are completed or all are rolled back), *consistency* (of the data), *isolation* (preventing one transaction from seeing any results of other transactions until those transactions are complete), and *durability* (ensuring that committed transactions are persisted even in the event of system failure after the commit operation). It is a convention to refer to these collectively by the acronym ACID.

A resource manager can support no transactions, *local transactions*, or *global transactions* (also known as *distributed transactions* or *XA transactions*). A global transaction is managed by an external transaction manager, such as through a Java Transaction API (JTA) implementation. A local transaction is typically managed internally within a single resource manager. (It can optionally can be managed by an external transaction manager, but this probably results in needless overhead.) Executing a transaction to a single resource manager, verifying the ACID properties, then committing the transaction is known as a *one-phase commit* protocol. To ensure complete data consistency and integrity, a transaction across multiple resource managers must use a *two-phase commit* protocol, consisting of a "prepare" phase and a "commit" phase. Updates are not committed to any resource manager unless all resource managers are prepared for updates.

All data manipulations that are part of a single transaction share a *transaction context*.

Relationship with Enterprise JavaBeans Technology

Enterprise JavaBeans technology, designed for data integrity and security, plays a large role in the overall transaction model of the J2EE platform. The J2EE Connector Architecture is tied closely to the EJB architecture, which is designed to handle transaction management for an application. A typical J2EE application executing transactions with one or more EISs will do so through EJB components.

In EJB technology, the demarcation (essentially, the beginning and end) of a transaction can be managed either by a bean, in a model known as *bean-managed transaction demarcation*, or by the container, in a model known as *container-managed transaction demarcation*. Entity beans, designed for data access, must use container-managed transaction demarcation. Session beans can use either model.

Transactions associated with J2CA connections are intended to seamlessly extend EJB transactions, regardless of which demarcation model is used. If an EJB method executes in the scope of a transaction, all work done by an EIS on behalf of the EJB is within the scope of the transaction, assuming the EIS supports global transactions. (Also see "OC4J Support for Last-Resource-Commit Optimization" on page 4-9 for information about how a single EIS that does not support global transactions can be included in a global transaction.)

See the *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* for details about EJBs in OC4J.

Relationship with Java Transaction API Technology

The Java Transaction API (JTA) is the transaction management vehicle of the J2EE platform. Any typical J2EE application performing global transactions, usually through EJBs and using resource adapters to access EISs, employs JTA. Use of the JTA involves a transaction manager that is external to the resource managers and

coordinates the overall transaction. The JTA is a set of interfaces between a transaction manager and the entities involved in a transaction—the application component, application server, and resource manager.

Important features provided by the JTA include two-phase commit, introduced earlier, which includes transaction recovery in the event that a resource manager fails during a transaction.

Key JTA interfaces, implemented by OC4J, include the `Transaction`, `TransactionManager`, and `UserTransaction` interfaces of the `javax.transaction` package. A `Transaction` object represents a transaction and has methods, such as `commit()` and `rollback()`, to allow an application to perform operations in the transaction. A `TransactionManager` object has methods allowing J2EE containers such as OC4J to manage the boundaries of a transaction, for container-managed transaction demarcation. A `UserTransaction` object has methods allowing an application to manage the boundaries of a transaction, for bean-managed transaction demarcation. An application component would use JNDI to look up a `UserTransaction` object. See "Implemented by Transaction Managers: UserTransaction and TransactionManager" on page 4-6 for additional information.

The JTA `javax.transaction.xa.XAResource` interface is implemented by resource adapter providers to represent resources, such as EISs, used in global transactions controlled by an external transaction manager. The transaction manager obtains an `XAResource` object for each EIS connection that is participating in the transaction, and it can associate and disassociate each resource with the transaction. See "Implemented by Resource Adapters: XAResource and LocalTransaction" on page 4-5 for additional information.

Refer to *Oracle Containers for J2EE Services Guide* for additional information about the Java Transaction API.

Local Transaction Management

A local transaction is one that is specific to a single resource manager, and is created and committed against this resource manager. The transaction can be demarcated by an application component or by the container.

Demarcation by an application component would involve EIS-specific APIs, such as, for example, methods of the CCI `LocalTransaction` interface (for a resource adapter that implements the CCI) or methods of the JDBC `java.sql.Connection` interface, such as `commit()` and `rollback()` (for a JDBC driver).

Demarcation by a container would involve methods of the SPI `LocalTransaction` interface.

Note: For container-managed transaction demarcation, where only one resource manager is involved in a JTA transaction, a container has two choices:

- Use its transaction manager to manage the transaction, using one-phase-commit optimization. This is what OC4J does. (See "Highlights of Required Transaction Support" on page 4-7.)
- Let the resource manager coordinate the transaction internally, using a local transaction of the resource manager without involving an external transaction manager.

Global Transaction Management

Global transactions typically use JTA `XAResource` objects, as noted earlier. This interface is based on the industry standard X/Open transaction model. The OC4J transaction manager coordinates all resource managers involved in the transaction.

For data integrity and consistency, `XAResource` objects support the two-phase-commit protocol to ensure that a transaction is committed successfully across all the resource managers, or is committed to none. In this protocol, the OC4J transaction manager calls the `XAResource prepare()` method on each resource manager to notify it of a pending commit. Only if each resource manager can be reached, and agrees to the commit, will the transaction manager proceed by calling the `XAResource commit()` method on each resource manager to commit the transaction. If, during the prepare phase, any resource manager cannot be reached or vetoes the commit, then the transaction manager aborts the transaction by calling the `XAResource rollback()` method on each resource manager.

Resource Adapter Configuration for Level of Transaction Support

In the `ra.xml` file supplied by a resource adapter provider, the level of transaction support of a resource adapter is indicated in the `<transaction-support>` element. According to version 1.5 of the *J2EE Connector Architecture Specification*, this is a subelement of an `<outbound-resourceadapter>` element and supports the values `NoTransaction`, `LocalTransaction`, or `XATransaction`, as in the following example for global transaction support:

```
<outbound-resourceadapter>
   ...
   <transaction-support>XATransaction</transaction-support>
   ...
</outbound-resourceadapter>
```

(In version 1.0 of the *J2EE Connector Architecture Specification*, `<transaction-support>` is a subelement of the `<resourceadapter>` element.)

Support levels are as follows:

- `NoTransaction` means the resource adapter supports no transactions at all, neither global transactions nor local transactions. Data can be written or updated, but no ACID properties are possible across multiple interactions with the EIS.
- `LocalTransaction` means the resource adapter supports local transactions, implementing the SPI `LocalTransaction` interface, but does not support global transactions.
- `XATransaction` means the resource adapter supports global transactions as well as local transactions, implementing the SPI `LocalTransaction` interface and the `XAResource` interface.

If the transaction support level of the resource adapter is `XATransaction`, then the J2CA container will call the `getXAResource()` method on the `ManagedConnection` instance when an `XAResource` object is needed to enlist the connection with a global transaction. If the transaction support level of the resource adapter is `LocalTransaction`, then the container will call the `getLocalTransaction()` method to get a `LocalTransaction` object. If the local transaction-level connection needs to be enlisted in a global transaction, then the container will create an emulated `XAResource`. The emulated `XAResource` gets enlisted and allows the local transaction-level connection to participate in a global transaction through the `LocalTransaction` object (with restrictions).

Overview of Key Interfaces Used in Transaction Management

The following sections summarize key J2CA and Java Transaction API interfaces that are used in transaction management:

Implemented by Resource Adapters: XAResource and LocalTransaction

Each of the following interfaces, implemented by resource adapter providers, plays an important role in the transaction management contract:

- `javax.transaction.xa.XAResource`
- `javax.resource.spi.LocalTransaction`

`XAResource` or `LocalTransaction` instances can be obtained from a managed connection, through methods specified in the SPI `ManagedConnection` interface. OC4J always obtains a `ManagedConnection` object from the resource adapter, through the resource adapter `ManagedConnectionFactory` object, when an application component requests a connection. This is regardless of whether work will be in the context of a transaction. The `ManagedConnection` object represents a physical EIS connection and may be either newly created or obtained from a connection pool. (The process of obtaining a managed connection is described in "Summary of the Connection Management Contract" on page 3-14.) Classes implementing `ManagedConnectionFactory` and `ManagedConnection` are supplied by the resource adapter provider.

The `ManagedConnection` interface specifies the following methods:

- `XAResource getXAResource()`
- `LocalTransaction getLocalTransaction()`

If the application component requesting a connection is in a global transaction context, and the resource adapter supports the `XATransaction` transaction support level, then OC4J calls the `getXAResource()` method to obtain an `XAResource` object from the managed connection. OC4J then enlists the `XAResource` object with the OC4J transaction manager. (The transaction manager obtains an `XAResource` object for each connection participating in a global transaction.) An `XAResource` object enables the EIS to participate in transactions that are controlled by an external transaction manager, such as the OC4J transaction manager. The transaction manager uses the `XAResource` object to communicate information about the transaction—such as association, completion, and recovery—to the EIS.

Key `XAResource` methods include the following:

- `void start(Xid, ...)` to start the work of a transaction branch
- `void end(Xid, ...)` to end the work of a transaction branch
- `void prepare(Xid, ...)` to execute the "prepare" phase of a two-phase commit
- `void commit(Xid, ...)` to execute the "commit" phase of a two-phase commit
- `void rollback(Xid, ...)` to roll back the transaction

Each of these takes a `javax.transaction.xa.Xid` instance to identify the transaction.

If the application component requesting a connection is in a local transaction context, or the application component is in a global transaction context but the resource adapter supports only the `LocalTransaction` transaction support level, then OC4J calls the `getLocalTransaction()` method to obtain a `LocalTransaction` object from the managed connection. A `LocalTransaction` object enables OC4J to manage a local transaction with the EIS.

The SPI `LocalTransaction` interface specifies three methods:

- `void begin()` to begin the local transaction
- `void commit()` to commit the local transaction
- `void rollback()` to roll back the local transaction

> **Note:** Do not confuse the SPI `LocalTransaction` interface, implemented by a resource adapter provider, with the CCI `LocalTransaction` interface, optionally implemented for client-side use. The CCI interface is for application-level local transaction demarcation. The SPI interface is used by the J2EE container for local transaction management. Both interfaces define the same methods, however.

Implemented by Transaction Managers: UserTransaction and TransactionManager

The following interfaces are implemented by transaction managers, such as the OC4J transaction manager:

- `javax.transaction.UserTransaction`
- `javax.transaction.TransactionManager`

The OC4J transaction manager allows global transactions to be supported across multiple EISs. A global transaction can be started either by an application component (typically an EJB) in a component-managed (bean-managed) scenario, or by OC4J in a container-managed scenario. In the former case, the component would use a `UserTransaction` object, which includes the following methods:

- `void begin()` to create a transaction and associate it with the current thread
- `void commit()` to commit the transaction associated with the current thread
- `void rollback()` to roll back the transaction associated with the current thread

> **Note:** In a global transaction scenario, where an application component has called the `begin()` method of a `UserTransaction` object (used for bean-managed transaction demarcation), OC4J enlists a resource manager `XAResource` object with the OC4J transaction manager whenever a connection handle for that resource manager is returned to the application. And OC4J disassociates the `XAResource` object from the transaction manager when the application component releases or closes the connection.

When a global transaction is started by OC4J in a container-managed scenario, OC4J manages the transaction through a `TransactionManager` object, which includes the following methods:

- `void begin()` to create a transaction and associate it with the current thread

- `void commit()` to commit the transaction associated with the current thread
- `void rollback()` to roll back the transaction associated with the current thread
- `Transaction suspend()` to suspend and return a transaction (`Transaction` object) that was associated with the current thread, and disassociate the resource from the transaction
- `void resume(Transaction)` to resume the association of the current thread with the transaction (represented by the `Transaction` object), and also reassociate the resource with the transaction

Once the transaction begins, and an application component requests a connection, the connection management mechanism is executed as described in "Summary of the Connection Management Contract" on page 3-14.

Implemented by OC4J: ConnectionEventListener Interface

OC4J implements the SPI `ConnectionEventListener` interface, which allows notifications from EISs regarding connection and transaction status, and registers each listener instance with the appropriate managed connection factory.

OC4J relies on notification when a connection is closed, to accomplish connection cleanup and any transaction cleanup required as a result of closing the connection. The following `ConnectionEventListener` method is called by the resource adapter:

- `void connectionClosed(ConnectionEvent)`

For local transactions, OC4J relies on notifications when a transaction is started, committed, or rolled back. The following `ConnectionEventListener` methods are called by the resource adapter:

- `void localTransactionStarted(ConnectionEvent)`
- `void localTransactionCommitted(ConnectionEvent)`
- `void localTransactionRolledback(ConnectionEvent)`

Each of these methods takes an SPI `ConnectionEvent` object that represents the event that OC4J is being notified of.

Transaction Support in the OC4J Connector Implementation

The following sections summarize highlights of the OC4J implementation of J2CA transaction management:

- Highlights of Required Transaction Support
- Highlights of Optional Transaction Support
- OC4J Configuration for Transaction Recovery

Highlights of Required Transaction Support

The following key OC4J features for resource adapters reflect requirements of the J2EE Connector Architecture:

- OC4J reads `<transaction-support>` elements of an `ra.xml` file to determine the level of transactions supported by the declared resource adapters.
- OC4J detects and prevents any occurrences of illegal transaction demarcation in an application component, such as an attempt to begin a global transaction while a local transaction is still in progress.

- As described earlier, OC4J supports the local transactions contract based on the SPI `LocalTransaction` interface, and supports global transactions that are based on functionality of the `XAResource` interface. (See "Implemented by Resource Adapters: XAResource and LocalTransaction" on page 4-5.)
- OC4J can use the one-phase-commit protocol for transactions using a single physical connection to a resource manager. *One-phase-commit optimization*, supported by OC4J, is for the scenario of a global transaction context where all logical connections in the transaction use the same physical connection to the same EIS, which must support at least one-phase-commit protocol. (Multiple logical connections may all point to the same physical connection due to connection sharing, for example, assuming all connections use the same data source and user name.) In this case, OC4J uses only the "commit" phase of the two-phase-commit protocol. The "prepare" phase is skipped.
- As noted earlier, OC4J implements the SPI `ConnectionEventListener` interface, and relies on notifications from the listener for local transaction starts, commits, and rollbacks. For a global transaction, OC4J relies on notifications for connection-closed events. (See "Implemented by OC4J: ConnectionEventListener Interface" on page 4-7.)
- OC4J detects incomplete local transactions. For example, if a stateless session bean starts a local transaction through a method invocation, but returns from the method without completing the transaction, then OC4J terminates the session and throws an exception.
- If an application component terminates due to an exception during a method invocation, OC4J resets all local transactions in progress and resets any client-specific state.
- OC4J supports connection sharing. See "Understanding Connection Sharing in OC4J" on page 4-12.

Highlights of Optional Transaction Support

This section discusses OC4J support for features that are optional or beyond the scope of the J2CA specification. The following topics are covered:

- OC4J Support for Lazy Enlistment of Connections in a Transaction
- OC4J Support for Last-Resource-Commit Optimization
- Unsupported Transaction Scenarios
- OC4J Checking for Illegal Transaction Demarcation

OC4J Support for Lazy Enlistment of Connections in a Transaction

Typically, when an application component requests a connection handle in the context of a transaction, the connection is automatically, or "eagerly", enlisted in the transaction. This is true even if the connection is ultimately unused, which results in unnecessary overhead.

With "lazy" enlistment, a new connection is enlisted in the transaction only if it is actually used in the transaction—in other words, only if data is transmitted through the connection.

Both a resource adapter and an application server must support lazy enlistment for this feature to be usable. The OC4J connection manager supports this feature through its implementation of the SPI `LazyEnlistableConnectionManager` interface, so you can use lazy enlistment with any resource adapter that implements the

`LazyEnlistableManagedConnection` interface for its managed connections. A managed connection calls the `lazyEnlist()` method on the connection manager, which results in enlistment of the connection only if it is used.

Both the resource adapter and OC4J are capable of determining whether the other implements the requisite interface. Neither attempts lazy enlistment if the other does not support it.

OC4J Support for Last-Resource-Commit Optimization

The OC4J 10.1.3 implementations of the Java Transaction API and J2EE Connector Architecture support *last-resource-commit optimization* for a situation where one (and only one) resource adapter supporting `LocalTransaction` (one-phase commits) is to be included in a global transaction with resource adapters supporting `XATransaction` (two-phase commits). Essentially, last resource commit optimization allows for a single non-XA-compliant resource to participate in an XA transaction, in the root OC4J process. If more than one non-XA-compliant resource is enlisted in the transaction, then an exception is thrown from the enlistment attempt.

For resource adapters, this optimization is accomplished by emulating an `XAResource` object, using an instance of the type `EmulatedXAResource` to represent the one-phase-commit resource adapter, with the resource adapter receiving notifications through the emulated XA resource. If there is any attempt to include a second resource adapter supporting only one-phase-commit protocol, the OC4J transaction manager throws an exception.

The transaction manager takes the following steps when it receives a request to commit a global transaction that includes one emulated XA resource:

1. It invokes a `prepare()` call on each of the two-phase-commit resources.
2. If all of the two-phase-commit resources are successfully prepared, then `commit()` (one-phase) is called on the emulated XA resource.
3. If the one-phase commit on the emulated XA resource completes successfully, then the transaction manager logs a decision to commit the global transaction and calls `commit()` on each of the two-phase-commit resources.

   Or else, if the one-phase commit on the emulated XA resource fails, then the transaction manager logs a decision to roll back the global transaction and calls rollback() on each of the two-phase-commit resources.

See "Last Resource Commit" under *Oracle Containers for J2EE Services Guide* for more information.

Notes Regarding Last-Resource-Commit Optimization Be aware of the following regarding last-resource-commit optimization:

- An emulated XA resource also implements the `SinglePhaseResource` interface, which is what signals its status as an emulated resource to OC4J.
- Last-resource-commit optimization becomes problematic if a failure occurs after `commit()` is called on the emulated XA resource but before the transaction manager receives the result of that call. In such a case, it is not possible to assure that the result of the global transaction will be atomic.
- With emulated XA resources, transaction recovery is not applicable.
- The transaction manager treats an emulated XA resource as a standard XA resource if logging is disabled in the transaction manager (in which case there can be no recovery).

Unsupported Transaction Scenarios

OC4J, in accordance with the J2EE Connector Architecture, does not generally support attempts to include multiple non-XA resource adapters within a single transaction. (Note that the discussion in the preceding section, "OC4J Support for Last-Resource-Commit Optimization", specifies a single non-XA resource adapter only.)

> **Note:** There are exceptions. If logging is disabled in the transaction manager, in which case recovery is not possible, then multiple one-phase-commit resources are allowed to participate in a transaction. Also, multiple emulated XA resources may be used in a transaction in testing scenarios, such as for performance testing.

Also, we do not recommend that you attempt a global transaction in which a resource adapter is at `XATransaction` support level but supports only one-phase commit (although a one-phase-commit resource can provide an `XAResource` object to emulate global transactions). The lack of true support for two-phase commit would not be known until it was time to commit the global transaction.

OC4J Checking for Illegal Transaction Demarcation

OC4J, through its implementation of the SPI `ConnectionEventListener` interface, flags cases of illegal transaction demarcation.

As an example, assume an EJB using bean-managed demarcation begins a local transaction (such as by using the CCI `LocalTransaction` interface, if that is supported by the resource adapter). Further assume that the EJB, before completing the local transaction, tries to begin a global transaction through a `begin()` call on a `UserTransaction` object. The resource adapter will have sent a "local transaction started" message to OC4J, but because OC4J will not yet have received a "local transaction committed" or "local transaction rolled back" notification at the point when the attempt is made to start a global transaction, OC4J will throw an exception on the `UserTransaction begin()` call.

OC4J Configuration for Transaction Recovery

Little configuration is required in OC4J for the transaction management contract. Functionality is largely behind the scenes. However, transaction recovery requires the configuration of a special user name and password. If a failure occurs during transaction processing, the OC4J recovery manager may need to contact an EIS, which will usually require special credentials, to inquire about any transactions that are in doubt. See "Understanding and Configuring Transaction Recovery" on page 4-15.

You can configure XA transaction recovery through the Application Server Control Console:

1. In the Connection Factories tab accessed from the Resource Adapter Home page for the appropriate resource adapter, choose the JNDI location of the connection factory that you want to configure.
2. In the Options tab of the resulting Edit Connection Factory page, you can do any of the following tasks:
   - Add a new user name.
   - Specify a password directly or indirectly, after specifying the user name.

     For a direct password, choose **Password** and type the password itself.

For an indirect password, choose **Indirect Password** and type a key (which might just be the user name, for example). OC4J uses the key to do a lookup in the User Manager (specifically, in the `jazn-data.xml` file).

- Change an existing user name or password.

OC4J Transaction Setup and Cleanup

The following sections describe how OC4J sets up before transactions and cleans up afterward, for the possible transaction levels and resource adapter support levels:

- Global Transaction Setup and Cleanup
- Local Transaction Setup and Cleanup

Global Transaction Setup and Cleanup

This section covers OC4J transaction setup and cleanup for global transactions, either where all involved resource adapters are at the `XATransaction` support level, or where last-resource-commit optimization is used, as discussed in "OC4J Support for Last-Resource-Commit Optimization" on page 4-9. (In the scenario of a global transaction with a resource adapter supporting only `NoTransaction`, OC4J completes connection setup as described in "Summary of the Connection Management Contract" on page 3-14, but does not perform any steps for transaction setup.)

If a connection allocation request is invoked from a J2EE application component that is within a global transaction context, and the resource adapter provides `XATransaction` support level (or there is a last-resource-commit scenario, as will be noted), OC4J performs the following steps to enlist the work done in the connection to the transaction manager:

1. Obtains a managed connection from the resource adapter—either a newly created managed connection or an existing one from the connection pool. See "Summary of the Connection Management Contract" on page 3-14 for details.
2. Obtains an `XAResource` object from the managed connection. The OC4J transaction manager will use the `XAResource` object to provide any transaction notifications to the resource manager.

   Note: For the resource adapter supporting only `LocalTransaction` in a last-resource-commit scenario, OC4J obtains, or if necessary creates, an `EmulatedXAResource` object associated with the managed connection, instead of obtaining an `XAResource` object directly from the managed connection. In this case, substitute `EmulatedXAResource` for `XAResource` in the steps that follow.

3. Enlists the `XAResource` object to the OC4J transaction manager.
4. Invokes the `start()` method of the `XAResource` object. For any work done through the connection, the resource manager is instructed to associate the work with a transaction branch.
5. Creates a connection handle from the managed connection and hands it back to the application component.

When the application component closes the connection handle, the resource adapter sends a `ConnectionClosed` event to OC4J. If there are no outstanding connection

handles for the underlying managed connection, OC4J calls the `end()` method of the `XAResource` object to inform the resource manager that any further operations on the managed connection will not be associated with the global transaction (performed by the OC4J transaction manager).

Local Transaction Setup and Cleanup

This section covers OC4J transaction setup and cleanup for local transactions, where all involved resource adapters are at either the `LocalTransaction` or `XATransaction` support level. (In the scenario of a local transaction with a resource adapter supporting only `NoTransaction`, OC4J completes connection setup as described in "Summary of the Connection Management Contract" on page 3-14, but does not perform any steps for transaction setup or cleanup.)

A local transaction can be started by OC4J, similarly to what is described in "Global Transaction Setup and Cleanup" on page 4-11. Alternatively, a J2EE application component can explicitly begin and end a local transaction through the use of an application-level transaction demarcation interface. The particular process used in starting a local transaction depends on the connection interface being used. For resource adapters that implement CCI as the client interface, an application component can obtain a CCI `LocalTransaction` object from a CCI `Connection` object, through the `getLocalTransaction()` method.

Regardless of the particulars of the client-side API, the following steps are executed between the resource adapter and OC4J when a local transaction is started:

1. The resource adapter detects the request from the application component to start the local transaction. For example, in CCI this is when the application component invokes the `begin()` method of the CCI `LocalTransaction` object.
2. The resource adapter notifies OC4J that a local transaction is starting. OC4J relies on this notification in order to provide appropriate services.
3. The resource adapter executes an EIS-specific process to start the local transaction.

Similarly, an application component can explicitly end a local transaction by using an API provided by the resource adapter. As is the case when starting a local transaction, which particular methods to use in ending a local transaction depends on the connection interface being used.

Here is what happens between the resource adapter and OC4J when a local transaction is ended:

1. The resource adapter detects the request from the application component to end the local transaction. For example, in CCI this is when the application component invokes the `commit()` or `rollback()` method of the CCI `LocalTransaction` object.
2. The resource adapter notifies OC4J that a local transaction has ended. OC4J will then update its internal states accordingly.
3. The resource adapter executes an EIS-specific process to commit or roll back the local transaction on the EIS.

Understanding Connection Sharing in OC4J

Connection sharing, specified in the J2EE Connector Architecture, allows multiple connection handles to be shared between multiple application components within a single transaction. This allows better performance and more efficient use of resources,

and helps avoid resource manager lock contentions and read-isolation problems. The following sections discuss this feature:

- Conditions for Connection Sharing
- Connection Sharing Scenario

Conditions for Connection Sharing

This section discusses conditions for connection sharing in OC4J:

- General Conditions for Connection Sharing
- Restriction on Connection Sharing for NoTransaction Support Level

General Conditions for Connection Sharing

When a new connection is requested by a J2EE application component, OC4J will attempt to return a new connection handle from an existing physical connection in the connection pool if the following conditions are satisfied:

- The connection request is for the same resource manager as is accessed through the existing physical connection, and uses the same data source.
- The connection request has the same properties as the physical connection, such as security attributes, isolation levels, character settings, and localization.
- The application component requesting the connection is in the same transaction scope as the application component currently using the physical connection.
- The application component has a "shareable" scope so that physical connections obtained from the resource manager can be shared between multiple logical connections. This is true by default, or can be explicitly specified under the applicable `<resource-ref>` element in the standard `ejb-jar.xml` or `web.xml` configuration file. In the following example, the application component can look up a connection factory from the JNDI location `eis/myEIS`, as specified in the `<res-ref-name>` element, for creating logical connection handles to the EIS. With the value of the `<res-sharing-scope>` element being declared as `shareable`, multiple connection handles in the application component can share a single physical connection, provided that the other conditions for connection sharing are met.

```
<resource-ref>
   <res-ref-name>eis/myEIS</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Application</res-auth>
   <res-sharing-scope>shareable</res-sharing-scope>
</resource-ref>
```

  A setting of `unshareable` disables sharing. Note that it is permissible for one or more components to declare connections from a resource to be shareable, while one or more other components declare connections from the same resource to be unshareable.

Notes:

- The use of a single shared connection, as opposed to multiple physical connections, is transparent to the application. Applications must not make assumptions about shared connections being used.
- Unless a resource manager is marked as unshareable, connections to it must be used in a shareable way, avoiding such actions as modifying security attributes, isolation levels, character settings, and localization.

Restriction on Connection Sharing for NoTransaction Support Level

Connection sharing is not supported for a resource adapter with transaction support level `NoTransaction`.

Connection Sharing Scenario

This section describes a sample scenario for connection sharing. Assume you have an application with two EJB session beans: EJB1 and EJB2. Both use the same resource adapter, capable of local transactions, to connect to the same EIS within a single local transaction. The EIS is configured to be shareable, or is by default. Because a local transaction is always associated with a single physical connection, EJB1 and EJB2 in this scenario share the same physical connection under the local transaction scope.

Consider the following actions:

1. The client application calls a method on EJB1 with no transaction context. The method implementation in EJB1 uses container-managed transaction demarcation and requests a connection to the EIS.
2. The connection is obtained, at which point OC4J starts a local transaction. This transaction is associated with the managed connection object from which the connection handle was obtained, where the managed connection object represents the physical connection. Work performed by EJB1 on the EIS through the connection is included in the local transaction context.
3. EJB1 calls a method on EJB2, and the method implementation in EJB2 requests a connection to the EIS.
4. OC4J returns a connection handle to EJB2 using the same managed connection object as for the connection requested by EJB1. OC4J also maintains, across the method call from EJB1 to EJB2, an association between this managed connection object and the local transaction context. Work performed by EJB2 on the EIS through the connection is included in the local transaction context. Note that connection state, such as any open cursors, can therefore be maintained across method calls between EJB1 and EJB2 in this scenario.

   Note that if EJB2 calls the `close()` method on the connection handle before its method called from EJB1 has completed, OC4J does not initiate any cleanup on the physical connection at this time. This is because there is still an incomplete local transaction associated with the physical connection.

   When EJB1 regains control, it can still use the connection handle, assuming it had not also called the `close()` method.
5. Now assume EJB1 eventually calls the `close()` method on its connection handle. Because there is still an incomplete local transaction associated with the physical

connection, OC4J does not initiate any cleanup on the physical connection at this time. It must still complete the local transaction process.

6. When the original method call from the client application completes, OC4J commits the local transaction.
7. The managed connection object does a cleanup of its local transaction state.
8. OC4J returns the physical connection to the pool

Understanding and Configuring Transaction Recovery

This section discusses OC4J support for two-phase-commit recovery for resource managers accessed through outbound connections.

The goal of transaction recovery is to maintain data integrity when data manipulation with one EIS fails during a multiple-EIS (global) transaction. Transaction recovery is therefore not applicable for local or one-phase-commit transactions.

The following topics are covered:

- Understanding XA Recovery in OC4J
- Configuring XA Recovery in OC4J

Understanding XA Recovery in OC4J

When OC4J halts unexpectedly or loses the connection to a resource manager during a transaction, it enters a recovery mode (depending on the state of the transaction) where it attempts steps to complete the transaction that was in progress at the time of the failure. Responsibility for recovery is delegated to the OC4J recovery manager. The application itself is not involved.

The first step in XA recovery is to determine the transaction state and then attempt to acquire the necessary `XAResource` objects if the transaction state mandates recovery. If recovery is necessary, OC4J then tries to reestablish connection to the EISs as soon as possible. This requires sign-on information—user name and password for applicable connection factories—in the following circumstances:

- The application uses component-managed sign-on, where the application provides sign-on information that is invisible to OC4J.
- The EIS requires transaction recovery to be accomplished only through a special, privileged user account that is different from the account that the J2EE application uses.

If the application uses container-managed sign-on and the EIS has no special sign-on requirements for recovery, then OC4J does not require recovery sign-on information.

Table 4–1 summarizes OC4J XA recovery scenarios.

Table 4–1 OC4J XA Recovery Scenarios

Sign-on Mode for Original Connection	Mode for Sign-on Credentials	XA Recovery Credentials
Component-managed (application supplies connection credentials)	OC4J is not aware of, and does not store, the user name and password for signing on to the EIS.	XA recovery credentials are according to your configuration. (See the next section, "Configuring XA Recovery in OC4J".) If you do not provide XA recovery sign-on configuration, OC4J attempts to connect to the EIS without a user name or password. If this fails, then transaction recovery fails.
Container-managed	OC4J determines the user name and password for connecting to the EIS. Based on the user that is signed on to the application (the OC4J user), OC4J applies security mapping to allow that user to obtain the user name and password for the EIS.	Again, XA recovery credentials are according to your configuration. If you do not provide XA recovery sign-on information, OC4J retrieves OC4J user information, uses the security mapping mechanism to determine the user name and password for the EIS, then connects to the EIS.

Once connection is reestablished, an `XAResource` object is obtained from the connection and is used to complete recovery.

Notes: Also be aware of the following:

- For XA recovery sign-on, OC4J supports resource adapters that accept password credentials. Resource adapters expecting other types of credentials cannot participate in OC4J XA recovery.
- After an OC4J instance fails, do not change any configuration for container-managed sign-on before restarting. This includes any changes that would be reflected in `<security-config>` settings in the `oc4j-ra.xml` file. It also includes the logic of any principal mapping modules used to support container-managed sign-on.
- Transaction recovery does not apply to emulated XA resources.

Configuring XA Recovery in OC4J

To configure the XA recovery sign-on parameters for a connection factory, take the following steps in the Application Server Control Console:

1. In the Connection Factories tab accessed from the Resource Adapter Home page for the appropriate resource adapter, choose the JNDI location of the connection factory you want to configure.
2. In the Options tab of the resulting Edit Connection Factory page, you can do any of the following:
   - Add a new user name. After specifying the user name, you can specify a password directly or indirectly. For a direct password, choose "Password" and type the password itself. For an indirect password, choose "Indirect Password" and type a key (which might just be the user name, for example). OC4J uses the key to do a lookup in the User Manager (specifically, in the `jazn-data.xml` file).
   - Change an existing user name or password.

Table 4–2 documents the sign-on properties, noting equivalent XML elements in the `oc4j-ra.xml` file. These are subelements of `<password-credential>`, which is a subelement of `<xa-recovery-config>`, which is a subelement of `<connector-factory>`.

Also see the context-sensitive topic "Edit Connection Factory Options Page" in the Application Server Control online help.

Table 4–2 XA Recovery Properties

Application Server Control Property	Corresponding XML Entity	Description
User	`<username>`	Desired user name for XA recovery sign-on through this connection factory.
Password	`<password>`	Password indicator for XA recovery sign-on through this connection factory. Either the password itself for a direct password, or a key for lookup in the User Manager for an indirect password.

Understanding J2CA Connection Wrapping

OC4J provides J2CA connection wrapping, a form of lazy enlistment of connections, to ensure that connections are properly enlisted with transactions. Connection wrapping supports connection association.

Connection Association

The J2EE Connector Architecture specifies an association of an application-level connection handle with a `ManagedConnection` (physical connection) instance. For each connection request, a connection handle is created from a `ManagedConnection` instance. The connection handle maintains an association with the underlying `ManagedConnection` instance.

When a J2EE component caches a connection handle across transaction boundaries, the container may need to reassociate the handle with a different physical connection to maintain transactional correctness. Without this re-association, it is possible that work done in one transaction could be mixed up with work done in different transaction.

The container uses connection association to ensure that the connection handle remains associated with a physical connection that is appropriate for the current transactional context and connection sharing scope. Through the `associateConnection` method of the `ManagedConnection` interface, the container can change the association of a connection handle with a managed connection instance. The `associateConnection` method dissociates the given connection handle from its associated `ManagedConnection` instance and reassociates the connection handle with itself.

A connection handle can be switched from one `ManagedConnection` instance to another. When a method is invoked on a connection handle, the container may need to transparently switch the handle to point to an appropriate shared `ManagedConnection` instance with the correct transaction scope. The J2EE Connector specification does not specify how a container determines that a transaction boundary has been crossed (when work is being done on a cached connection in a new transaction context).

> **Note:** In most cases, switching `ManagedConnection` instances for connection handles used across transaction boundaries is required only if the connections are shareable. An application can indicate that the connections that it references are unshareable. A connection that is marked as unshareable can be referenced only by a single connection handle. This means that as long as connection handle for an unsharable connection is open, it can remain associated with the same ManagedConnection. This is true even if the connection handle is cached across transaction boundaries.

The J2CA lazy connection association optimization enables the J2CA container to reclaim hibernating `ManagedConnection` instances, which allows for a more optimal use of connection resources. This is achieved by allowing the J2CA container to dissociate a connection handle from its associated `ManagedConnection` instance, leaving the handle in an inactive state and freeing the `ManagedConnection` instance for reuse. The next time the connection handle is used, it needs to be reassociated with an appropriate `ManagedConnection` instance. With lazy association, it is up to the resource adapter to notify the J2CA container when a connection handle needs to be re associated if the handle has been dissociated from its `ManagedConnection` instance.

> **Note:** The lazy connection association is optional for the resource adapter.

Transaction Enlistment

In most cases the connection association mechanism will ensure that a connection handle is associated with a `ManagedConnection` instance that is appropriate for the current transaction context. However, in some cases where a connection handle is used across a transaction boundary, connection reassociation is not appropriate, as in unsharable connections. In these cases, the J2CA container still needs to ensure that the currently associated `ManagedConnection` instance is enlisted in the current transaction context. As with connection association, the problem for the container is how to know that a transaction boundary has been crossed.

The J2CA lazy transaction enlistment optimization addresses this problem, since it is up to the resource adapter to notify the container that a connection should be enlisted in the transaction (if there is one). However, lazy transaction enlistment is optional. OC4J provides J2CA connection wrapping to ensure that connections are properly enlisted if a resource adapter does not support lazy enlistment.

J2CA Connection Handle Wrapping

When an application component obtains a connection through a J2CA resource adapter, OC4J creates a wrapper object around the connection handle returned by the connection factory. The wrapper object is what is actually returned to the application. By wrapping connections supplied by the resource adapter, the container can intervene to ensure that the connection handle is associated with a physical connection that is appropriate for the current transactional context and connection sharing scope.

OC4J wraps all connection handles with connection-handle proxies to perform connection association. You can cast a connection handle only to interfaces that the connection handle implemented. An attempt to cast a connection handle to a concrete class will cause a ClassCastException exception.

The connector connection wrapper classes implement the same interfaces as the resource adapter's connection handle. Each wrapper object is created with a reference to the actual connection handle object returned by the resource adapter. Every method call is intercepted in the wrapper before it is delegated to the underlying referenced connection handle.

For example, a call to some method on a connector connection wrapper would "intercept" and ensure that the connection handle was associated with a managed connection that was properly enlisted with the current transaction context. This may involve reassociating the connection handle to a new managed connection or simply making sure that the currently associated managed connection is enlisted in the current transaction context. The connector connection wrapper would then delegate the method call to the underlying connection handle object.

Connection Association Scenarios

The scenarios in this section illustrate some issues with using connection associations:

- Scenario 1: Enlistment of a Connection Obtained Outside a Transaction
- Scenario 2: Transactional Context Switch
- Scenario 3: Handles for a Shared Connection in Different Transactional Contexts

Scenario 1: Enlistment of a Connection Obtained Outside a Transaction

An application obtains a connection outside of a transaction and then use the connection in the scope of a transaction.

1. An application obtains a connection from Oracle J2CA.
2. The application begins a transaction using `UserTransaction.begin()`.
3. The application uses another resource in the transaction that gets enlisted.
4. The application uses the J2CA connection in the scope of the transaction.

The J2CA connection would not be associated with the transaction because it was created outside of the scope of the transaction. This will result in a nonatomic transaction outcome in which the work that is enlisted in the transaction might roll back and the work done on the J2CA connection could commit, or vice versa.

Scenario 2: Transactional Context Switch

An application obtains an emulated J2CA connection in the scope of a transaction so that the connection is enlisted and later used in another transactional context.

- An application starts a global transaction.
- The application calls a method on a stateful session bean with a required `tx` attribute. Because a transaction is already associated with the thread, the EJB container does not start a new transaction.
- An emulated JC2A connection is obtained in this method and used to alter some data. When created, the connection is enlisted with the transaction and cached by the instance (not closed).
- The method returns without committing the transaction because it was in place prior to the method invocation.
- The application calls a different method on the same stateful session bean with a `tx` attribute of `RequiresNew`. This forces the EJB container to suspend the

existing transaction association and start a new transaction for the method invocation.

- The bean uses the same connection handle that was used in the previous method invocation.

At this point, because the connection handle is still using the same 1pc physical connection as it was for the first method invocation, which was in a different transactional scope, all work done in method invocation two on the emulated J2CA connection is done on behalf of the original, suspended transaction and not the new transaction used in method invocation two. The connection handle must recognize that the transactional context has changes and swap out the physical connection being used by the handle.

This scenario will lead to incorrect outcomes because the second transaction may result in a rollback where the first may commit. In an attempt to prevent an inconsistent outcome, Oracle J2CA currently throws an exception on suspension of the transactional context in which an emulated connection is enlisted. This would prevent OC4J users from performing the preceding legal scenario.

Note: This scenario is a problem only for emulated connections because XA connections would send the begin and end calls to the XA resource and the resource would associate or disassociate the connection from a transaction. With an emulated connection, OC4J cannot send these begin and end calls to the resource, so OC4J must keep an association of physical connections to transactional contexts and switch physical connections according to the transaction context in place for any given invocation.

Scenario 3: Handles for a Shared Connection in Different Transactional Contexts

Two connection handles pointing to the same physical connection due to connection sharing are later used in different transactional contexts. This scenario may use either emulated or nonemulated connections.

1. An application calls a method on EJB A with a `tx` attribute of `Required`. The EJB container starts a new transaction.
2. EJB A gets a connection to `resource1` and uses the connection in the scope of the transaction. The connection is therefore enlisted with the transaction. The connection is cached for later use.
3. EJB A calls a method on EJB B, which has a `tx` attribute of `Required`. Because a transaction is already associated with the thread, a new transaction is not created.
4. EJB B also gets a connection to `resource1` and caches the connection. Because of connection sharing, both EJB A's connection handle and EJB B's connection handle will point to the same physical connection.
5. Both method invocations complete successfully and the transaction commits. At this point the transaction has committed and both connection handles point to the same physical connection.
6. Later, the application calls a different method on EJB A with a `tx` attribute of `Required`. The EJB container starts a new transaction.
7. EJB A uses the previously cached connection in the scope of the newly created transaction.

> **Note:** Currently, the connection is not associated with the transaction as in Scenario 1. This scenario is assuming that the enlistment issue has been fixed and the connection is properly enlisted in the transaction.

8. At the same time that EJB A is handling the preceding method invocation, another application thread calls a method on EJB B with a `tx` attribute of `Never`. Because of the `tx` attribute `Never`, the method will be invoked in a null transactional context.
9. EJB B uses the previously cached connection while not participating in a transaction.

At this point, EJB A and EJB B are using two distinct connection handles that point to the same physical connection. The physical connection is enlisted in EJB A's transaction but is also being used by EJB B with no transactional association. Because the connection is enlisted in EJB A's transaction, all work done by EJB B on its connection handle becomes part of EJB A's transaction, which is obviously incorrect.

This case is different than Scenario 2 in that it is valid for both XA connections and emulated connections. Also, the mechanism described in Scenario 3, where an exception is thrown from an emulated connection when suspend occurs, will not work in preventing an incorrect outcome in this case.

5

Work Management

This chapter, organized as follows, provides an overview of the J2CA work management contract and focuses on how to use the OC4J work management thread pool implementation:

- Overview of the Work Management Contract
- Using the OC4J Work Management Thread Pool

Overview of the Work Management Contract

This section discusses the basic essentials of the J2CA work management contract, covering the following topics:

- Understanding the Need for the Work Management Contract
- Introducing the Work Management Model and Key APIs

Understanding the Need for the Work Management Contract

While some resource adapters may be relatively simple, passively executing in the context of a single application thread, a typical resource adapter is multitasking and requires multiple threads to perform all its duties. These duties may include listening to a network endpoint or communicating with a network peer, in addition to performing internal work or calling application components.

It is possible for a resource adapter to create its own threads; however, because an application server is designed to manage all system resources and knows the overall state of the runtime environment, it is advantageous to allow the application server to create and manage the threads instead. This is the best way to ensure optimal system efficiency, manageability, and security. (For example, an application server may maintain a pool of threads to be shared by all deployed resource adapters.) In fact, an application server may prevent a resource manager from managing threads.

The purpose of the J2CA work management contract, between an application server and a resource adapter, is to provide a mechanism for resource adapters to use threads that are created and managed by the application server. Under the contract, a resource adapter submits work units to an application server for execution, and the application server work manager uses threads under its management to perform the work.

Introducing the Work Management Model and Key APIs

Interfaces and classes to implement the J2CA work management model are in the `javax.resource.spi.work` package.

A resource adapter implements the `Work` interface, instances of which represent units of work that the resource adapter submits to the application server for processing. The `Work` interface inherits a `run()` method, executed to perform the work, and specifies a `release()` method, which the application server may use to indicate that work should be completed as soon as possible.

An application server, such as OC4J, implements the `WorkManager` interface in order to manage work units. This interface specifies methods to do work (blocking until the work is completed), start work (blocking only until the work is started), and schedule work (returning as soon as the work unit has been accepted for processing). Each of these methods takes a `Work` instance as input. The `doWork()` method is for synchronous work, while the `scheduleWork()` method is for asynchronous work. The `startWork()` method guarantees first-in/first-out execution, and the work manager can optionally keep track of elapsed time between the acceptance and beginning of execution of a work unit.

A resource adapter obtains a `WorkManager` instance from the application server, creates `Work` instances for the units of work to be accomplished, and passes the `Work` instances to the `WorkManager` instance when it calls any of the methods to accomplish work. Optionally, a resource adapter may also pass the following to any of the `WorkManager` methods:

- An `ExecutionContext` instance, which models a context in which to execute the work unit, such as for transaction and security functionality.
- A `WorkListener` instance, which provides a callback event listener that is notified when work processing events occur. The notification mechanism includes methods, called by the application server, to indicate that work has been accepted, rejected, started, or completed.

The work manager may throw two types of work management exceptions: `WorkCompletedException`, indicating that a submitted `Work` instance completed but with an exception, or `WorkRejectedException`, indicating that a submitted `Work` instance was rejected. These exceptions can also be passed by the work manager to the resource adapter through the `WorkListener` mechanism, which is especially useful with asynchronous work submissions. Each of the `WorkListener` methods `workAccepted()`, `workRejected()`, `workStarted()`, and `workCompleted()` takes a `WorkEvent` instance as input. When a `WorkEvent` instance is created, it takes a `Work` instance and applicable work exception instance as input. (`WorkCompletedException` and `WorkRejectedException` are subclasses of `WorkException`, which is a subclass of `javax.resource.ResourceException`.)

The application server maintains a pool of threads to be used for work management. When a `Work` instance is submitted, it is picked up by a free thread for execution. The thread sets up an appropriate execution context and calls the `run()` method to perform the work. When work is complete, the application server is free to reuse the thread. Alternatively, depending on load circumstances, the application server may at any time call a `Work` instance `release()` method, from a separate thread, to try to reclaim the thread. This does not mandate any action by the resource adapter, but hints that it should release the active thread executing the work. The resource adapter should use threads carefully, monitor such hints, and perform any appropriate cleanup.

Each application server provides its own thread pool implementation, although all threads used in executing work units must be of the same priority level.

Using the OC4J Work Management Thread Pool

OC4J provides a configurable global thread pool specifically for use by deployed resource adapters. The following sections give an overview and discuss OC4J configuration and metrics features:

Overview of the Work Management Thread Pool

OC4J supports different types of thread pools, with one of them being specifically for use by resource adapters for work management. The work management thread pool operates independently of the others, but each pool supports the same basic set of features, being configurable for the following:

- Minimum number of threads that must exist in the pool
- Maximum number of threads that can execute simultaneously in the pool
- Maximum number of requests that can be in the queue waiting for a thread
- Time (in milliseconds) to keep a thread alive waiting for a request to service, after which the thread is destroyed

If the number of threads in the pool is under the minimum, OC4J will add a new thread rather than put a request in the queue. Otherwise, OC4J will queue the request rather than adding a new thread. OC4J generally attempts to keep the number of threads at or near the minimum, unless the queue is full.

See the *Oracle Containers for J2EE Configuration and Administration Guide* for more information about OC4J thread pools.

Configuring the Work Management Thread Pool

OC4J thread pools, including the thread pool for resource adapter work management, are configured in the OC4J `server.xml` file, through subelements of the top-level `<application-server>` element. You can configure the work management thread pool through the `<work-manager-thread-pool>` subelement or through Application Server Control.

In the Application Server Control Console, you can update attributes of the work management MBean, which is accessible through the System MBean Browser:

1. From the OC4J Home page, select the Administration tab.
2. From the Administration tab, under JMX, go to the System MBean Browser task.

   The System MBean Browser lists the system MBeans on the left and information about the selected MBean on the right.
3. Under ThreadPool in the left frame, select the WorkManager MBean; for example, WorkManager_0.

   The Attributes tab lists the attributes of the work management thread pool, with descriptions and values. You can edit the values of attributes that have RW (read/write) access.
4. Apply any changes.

Notes:

- The **Apply Changes** button will be visible only if the browser page contains at least one attribute with a modifiable value.
- If you make changes to properties of the work management thread pool through the Application Server Control Console, the changes will persist to the `server.xml` file only if a `<work-manager-thread-pool>` subelement is defined.

Alternatively, you can use the `<work-manager-thread-pool>` subelement of `<application-server>` to configure the work management thread pool. (Other OC4J thread pools are configured through the `<global-thread-pool>` element.) The `<work-manager-thread-pool>` attributes are documented in Table 5–1.

Table 5–1 Configuration Attributes for Work Management Thread Pool

Parameter	Description	Default Value
min	Minimum number of threads to exist in the pool	1
max	Maximum number of threads to execute simultaneously in the pool	`Integer.MAX_VALUE` (essentially, no maximum)
queue	Maximum number of requests (`Work` instances) in the queue, waiting for a thread	0
keepAlive	Number of milliseconds to keep a thread alive while it waits for a request to service	600,000 (10 minutes)
debug	Flag to enable output of diagnostic information for the thread pool	false

Note: The WorkManager thread pool uses three Work Manager threads for internal use. For example, if you specify `max="16"`, then only 13 work instance threads are available to service requests. Similarly, if the `max` value is `20`, then only 17 instances would be available. So you need to set `max` to your required maximum size plus 3.

The following example enables the debug flag and declares a thread pool with a minimum of 10 threads, a maximum of 100 executing threads, a queue size of 200 requests, and a keep-alive time of 60 seconds:

```
<application-server ... >
   ...
   <work-manager-thread-pool min="10" max="100" queue="200"
                             keepAlive="60000" debug="true" />
   ...
</application-server>
```

See the *Oracle Containers for J2EE Configuration and Administration Guide* for more information about configuring OC4J thread pools in the `server.xml` file.

Using Metrics for the Work Management Thread Pool

Dynamic Monitoring Service (DMS) metrics for the resource adapter work management thread pool are under `/oc4j/WorkManagementPool` in the DMS

hierarchy. There are two categories of metrics for the thread pool, one indicating thread pool configuration values as shown in Table 5–2, and the other indicating thread pool execution state as shown in Table 5–3.

Note: DMS adds performance-monitoring features to a number of Oracle Application Server components, including OC4J. The goal of DMS is to provide information about runtime behavior through built-in performance measurements so that users can diagnose, analyze, and debug any performance problems. DMS provides this information in a package that can be used at any time, including during live deployment. Data are published through HTTP and can be viewed with a browser. For general information about how to use DMS, the built-in DMS metrics that are available, and other OC4J performance considerations, refer to the *Oracle Application Server Performance Guide*.

Table 5–2 DMS Metrics for Work Management Thread Pool Configuration

Metric	Description
minPoolSize	Indicates the minimum number of threads to exist in the pool. This corresponds to the `wm-min` configuration attribute.
maxPoolSize	Indicates the maximum number of threads to execute simultaneously in the pool. This corresponds to the `wm-max` configuration attribute.
maxQueueSize	Indicates the maximum number of requests (`Work` instances) in the queue, waiting to be serviced by a thread. This corresponds to the `wm-queue` configuration attribute.
keepAlive	Indicates the number of milliseconds to keep a thread alive while it waits for a request to service. This corresponds to the `wm-keepAlive` configuration attribute.

Table 5–3 DMS Metrics for Work Management Thread Pool State

Metric	Description
totalThreadCount	Indicates the total number of threads in the pool.
idleThreadCount	Indicates the number of threads in the pool that are waiting for a request to service.
queueSize	Indicates the number of requests (`Work` instances) waiting in the queue for a thread to become available.
queueFullEvent	Indicates the number of times that a request was rejected due to a full queue.
workStartDuration	Measures the duration between when a request is accepted and when a thread is allocated to complete the work. (If a thread is readily available, this would measure the processing overhead of the thread pool in finding an available thread and setting up the proper execution context for processing the work. If all available threads are busy handling other requests, this time would also include the queuing time.) Three separate values are expressed for this metric—minimum duration, maximum duration, and average duration.

The following example shows sample DMS output for the work management thread pool:

```
<statistics>
  /oc4j [type=n/a]
   /oc4j/Work_Management_Pool [type=oc4j_workManagementPool]
    idleThreadCount.level:       NOTIFICATION
    idleThreadCount.value:       4       threads
    keepAlive.level:     NOTIFICATION
    keepAlive.value:     600000  milliseconds
    maxPoolSize.level:   NOTIFICATION
    maxPoolSize.value:   20      threads
    maxQueueSize.level:  NOTIFICATION
    maxQueueSize.value:  50      work_requests
    minPoolSize.level:   NOTIFICATION
    minPoolSize.value:   5       threads
    queueFullEvent.count:        0       ops
    queueFullEvent.level:        NOTIFICATION
    queueSize.level:     NOTIFICATION
    queueSize.value:     0       work_requests
    totalThreadCount.level:      NOTIFICATION
    totalThreadCount.value:      5       threads
    workStartDuration.avg:       6.900943396226415       msecs
    workStartDuration.completed: 424     ops
    workStartDuration.level:     NOTIFICATION
    workStartDuration.maxTime:   34      msecs
    workStartDuration.minTime:   0       msecs
    workStartDuration.time:      2926    msecs
 </statistics>
```

6

Using RAs for Inbound Communication

This chapter describes concepts and configuration for using a resource adapter to allow an EIS to initiate communication to a message endpoint in the application server; specifically, to a message-driven bean (MDB). The following topics are covered:

Concepts: Using Resource Adapters for Inbound Communication

The following sections cover initial concepts in using resource adapters for inbound communication:

Introduction to Using Resource Adapters for Inbound Communication

This chapter considers a situation where an EIS must initiate communication with a J2EE application component to perform specific work. Such a scenario requires a resource adapter as the communication vehicle for the EIS, the optional use of a messaging system such as JMS (for example), and an MDB as the J2EE component that will receive the communication message. The MDB, in turn, may invoke an entity bean or session bean or use other resources to perform the work.

This type of scenario is possible through the J2CA inbound communication model, introduced in "Outbound Versus Inbound Communication Through Resource Adapters" on page 1-5. To support inbound communication, the J2CA message inflow contract provides a mechanism for invoking MDBs from a resource adapter. This and other related contracts are briefly introduced in "Overview of Related Contracts for Inbound Communication" on page 6-2.

This chapter describes configuration and other relevant actions relating to the following:

- Required configuration of the resource adapter
- Deploying and configuring the MDB and associating it with the resource adapter

- Message delivery lifecycle—activation of the MDB (the process where the MDB informs the resource adapter that it wants to receive a certain type of message), message delivery to the MDB, and deactivation of the MDB (the process where the MDB informs the resource adapter that it no longer wants to receive messages)
- Special conditions in message delivery—concurrent delivery, exceptions from MDBs, and failure during delivery within a transaction

Background and Use Case for Resource Adapters as Message Listeners for MDBs

In the EJB 2.0 specification, only J2EE containers (such as OC4J) could act as message listeners for MDBs, and MDBs were required to implement the `javax.jms.MessageListener` interface. In the EJB 2.1 specification, it is possible for a resource adapter to act as a message listener for MDBs, and an MDB can implement any message listener interface.

A resource adapter can support any type of listener, and any MDB can be associated with any resource adapter if they support the same type of listener. This relieves the J2EE container of potentially having to support an unlimited number of message listener types. In this way, the message listening capacity has become portable across containers.

In Oracle Application Server 10*g* Release 3 (10.1.3.1.0), to support JMS messaging implementations, Oracle supplies a generic JMS resource adapter that is the recommended vehicle for plugging JMS providers into OC4J. This generic adapter supports inbound communication (discussed in the next section, "Overview of Related Contracts for Inbound Communication"), but works only with MDBs implementing the `MessageListener` interface.

See "Introducing Oracle JMS Support and Generic JMS Resource Adapter" on page 1-12 for additional information about the generic JMS adapter.

Overview of Related Contracts for Inbound Communication

This section briefly discusses some of the J2CA contracts that are relevant to using resource adapters as message listeners for MDBs:

- Message inflow contract
- Transaction inflow contract
- Work management contract

The message inflow and transaction inflow contracts are not discussed in any detail in the current release of this manual, but OC4J simply implements the standard. The work management contract and key APIs are discussed in "Overview of the Work Management Contract" on page 5-1, but not in great detail. Refer to version 1.5 of the *J2EE Connector Architecture Specification* for further information about these contracts.

Introduction to the Message Inflow Contract

The J2CA message inflow contract outlines a contract between a J2EE container and a resource adapter to allow delivery of messages through the resource adapter to an MDB, where both execute in the J2EE container. The MDB must be configured to indicate what types of messages it wants to receive, and the resource adapter must be configured to indicate that it can support (seek, find, and relay) those types of messages.

Message delivery is typically asynchronous, but it is also possible for an MDB to send and receive messages synchronously, through APIs specific to a particular message system.

Introduction to Imported Transactions and the Transaction Inflow Contract

There are many circumstances where a message provider creates a transaction for message delivery and sends messages to the resource adapter within that transaction. In this situation, the resource adapter imports the transaction and attempts to relay the messages to the MDB within the same transaction. The specific actions of the resource adapter for this exchange are according to the J2CA transaction inflow contract.

This contract, between a J2EE container and a resource adapter, allows the resource adapter to propagate the transaction to the J2EE container so that work can be performed there as part of the imported transaction. The contract also allows a resource adapter to accept transaction completion and recovery calls initiated by an EIS, so that ACID properties of the transaction are maintained. (See "Transaction Characteristics and Scopes" on page 4-2 regarding ACID properties.)

Note that when messages are delivered within the context of an imported transaction, the application server must take that into account when handling transaction demarcation.

Relationship Between Imported Transactions and the Work Management Contract

When a transaction, started by an external transaction manager, is imported by a resource adapter, the resource adapter uses the J2CA work management contract to introduce to the J2EE container an execution context object that includes identification information for the transaction. The J2EE container must then set up a transaction context internally that is visible to other components executing in the same thread of control.

The purpose of the work management contract, between the J2EE container and the resource adapter, is to provide a mechanism for a resource adapter to perform work by submitting work units to the J2EE container for execution. The contract allows the resource adapter to schedule work, such as message delivery to an MDB, to be performed by container-managed threads.

Configuration and Deployment to Use an RA as a Listener for an MDB

Using a resource adapter as a message listener on behalf of an MDB requires the following:

- A J2EE-compliant application server that supports version 1.5 of the *J2EE Connector Architecture Specification*, as is the case with OC4J
- A resource adapter that is deployed on the application server, is capable of message delivery in general, and supports the message types the MDB wants to receive in particular
- Deployment of the MDB to the application server

Notes:

- The resource adapter is typically provided by a third party to plug in a particular kind of message system to the application server. It is often standalone, and that situation is implied by some of the discussion in this chapter, but it can alternatively be deployed in the EAR file with the application that uses it.
- More than one application, or more than one MDB within an application, can use the same resource adapter as a message listener.
- An MDB can be undeployed and redeployed any number of times during the lifecycle of a resource adapter; however, a resource adapter cannot be undeployed until all MDBs activated against it have been undeployed.

The rest of this section discusses the following topics:

Understanding the Resource Adapter Configuration

To support an MDB for message listening, OC4J must have a deployed resource adapter that supports the message system and message listener type that the MDB will use. The resource adapter is often a standalone adapter previously deployed, but can also be deployed with the same application as the MDB (in the same EAR file).

Resource adapter configuration to support MDBs is included in the standard `ra.xml` file, which lists the message listener types that the resource adapter supports. Each listener type is specified as a fully qualified Java type that indicates the message listener interface that is implemented.

Because MDB support requires an inbound resource adapter, the resource adapter is configured in an `<inbound-resourceadapter>` element. This involves a `<messageadapter>` subelement, one or more `<messagelistener>` subelements of `<messageadapter>` (one for each type of message listener supported by the resource adapter), and the following subelements of `<messagelistener>`:

- The `<messagelistener-type>` subelement indicates the type of message listener to be used and must match a message listener type implemented by the MDB, as specified in a `<messaging-type>` element of the `ejb-jar.xml` file where the MDB is configured.
- The `<activationspec>` subelement specifies information that OC4J must have in order to activate the MDB at runtime, as follows:
  - For each message listener type, the resource adapter provides a JavaBean class that implements the SPI `ActivationSpec` interface, and the name of this class is indicated in the `<activationspec-class>` subelement of `<activationspec>`. An instance of this class contains activation information

for a message endpoint (an MDB) and is passed to the resource adapter during endpoint activation.

- Any bean properties of this class that must be set for activation are indicated under the `<required-config-property>` subelement of `<activationspec>`, with a `<config-property-name>` element for each. Each `<config-property-name>` specification corresponds to a matching `<activation-config-property>` name/value pair in the `ejb-jar.xml` file.

The following example is for a resource adapter that supports a Java Message Service (JMS) message system, using the `javax.jms.MessageListener` listener type. The resource adapter activation spec implementation is the JavaBean `oracle.j2ee.ra.jms.generic.JMSActivationSpec`, which has three properties that must be set for activation: `ConnectionFactoryJndiName`, `DestinationName`, and `DestinationType`.

```
<connector ... >
   ...
   <resourceadapter>
      ...
      <inbound-resourceadapter>
         <messageadapter>
            <messagelistener>
               <messagelistener-type>
                  javax.jms.MessageListener
               </messagelistener-type>
               <activationspec>
                  <activationspec-class>
                     oracle.j2ee.ra.jms.generic.JMSActivationSpec
                  </activationspec-class>
                  <required-config-property>
                     <config-property-name>
                        ConnectionFactoryJndiName
                     </config-property-name>
                     <config-property-name>
                        DestinationName
                     </config-property-name>
                     <config-property-name>
                        DestinationType
                     </config-property-name>
                  </required-config-property>
               </activationspec>
            </messagelistener>
            ...
         </messageadapter>
      </inbound-resourceadapter>
      ...
   </resourceadapter>
   ...
</connector>
```

Notes:

- A resource adapter can support multiple message listener types, so there can be multiple <messagelistener> elements for each <inbound-resourceadapter> element in the ra.xml file. However, each <messagelistener> element must specify a unique message listener type.
- In the Application Server Control, the supported message listener types for a resource adapter are indicated in the Resource Adapter Home page.
- For each <messagelistener> element, there is only one <activationspec> element.
- A resource adapter can support multiple activations for any given message listener type.

Understanding the MDB Configuration for Deployment

The MDB developer or deployer configures the MDB in the ejb-jar.xml file, through a <message-driven> element and the following subelements:

- The <messaging-type> subelement specifies the message listener interface that the MDB implements, and must match the message listener type specified in a <messagelistener-type> element in the ra.xml file of the resource adapter being used.
- The <activation-config> subelement and its <activation-config-property> subelements are for configuration properties specific to the message system that will be used—specifically, for customized property settings for the activation configuration JavaBean supplied by the resource adapter. The configuration is expressed in terms of name/value pairs specified in subelements of <activation-config-property>. The property names here correspond to property names from <config-property-name> elements in the ra.xml file and specify default values for at least the properties that are specified in ra.xml as being required. (What properties are recognized for a particular message system is beyond the scope of the J2CA specification or EJB specification.)

For complete information about configuring an MDB, refer to the *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* or to version 2.1 of the *Enterprise JavaBeans Specification*.

The example that follows is for use of queues with a JMS message system. To understand the property names and values in this example, you must have some knowledge of JMS.

```
<ejb-jar>
   ...
   <enterprise-beans>
      ...
      <message-driven>
         <ejb-name>simpleMdb</ejb-name>
         <ejb-class>oracle.j2ee.ra.jms.mqjratest.simpleMdb</ejb-class>
         <messaging-type>javax.jms.MessageListener</messaging-type>
         <transaction-type>Container</transaction-type>
         <activation-config>
            <activation-config-property>
              <activation-config-property-name>
```

```
                DestinationType
              </activation-config-property-name>
              <activation-config-property-value>
                javax.jms.Queue
              </activation-config-property-value>
            </activation-config-property>
            <activation-config-property>
              <activation-config-property-name>
                DestinationName
              </activation-config-property-name>
              <activation-config-property-value>
                java:comp/resource/MQSeries/MQQ
              </activation-config-property-value>
            </activation-config-property>
            <activation-config-property>
              <activation-config-property-name>
                MessageSelector
              </activation-config-property-name>
              <activation-config-property-value>
                RECIPIENT='MDB'
              </activation-config-property-value>
            </activation-config-property>
            <activation-config-property>
              <activation-config-property-name>
                ConnectionFactoryJndiName
              </activation-config-property-name>
              <activation-config-property-value>
                java:comp/resource/MQSeries/MQXAQCF
              </activation-config-property-value>
            </activation-config-property>
          </activation-config>
          ...
          <resource-ref>
             <description>connfactory for reply</description>
             <res-ref-name>jms/XAQCF</res-ref-name>
             <res-type>javax.jms.XAQueueConnectionFactory</res-type>
             <res-auth>Container</res-auth>
             <res-sharing-scope>Shareable</res-sharing-scope>
          </resource-ref>
          ...
          <resource-env-ref>
             <description>Queue to send reply to</description>
             <resource-env-ref-name>jms/QUEUE1</resource-env-ref-name>
             <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
          </resource-env-ref>
          ...
    </message-driven>
    ...
</enterprise-beans>
...
<assembly-descriptor>
    <container-transaction>
       <method>
          <ejb-name>simpleMdb</ejb-name>
          <method-name>onMessage</method-name>
          <method-params>
             <method-param>javax.jms.Message</method-param>
          </method-params>
       </method>
       <trans-attribute>Required</trans-attribute>
```

```
      </container-transaction>
      ...
   </assembly-descriptor>
   ...
</ejb-jar>
```

Notes:

- This sample also shows additional elements that are relevant for a messaging system example. The `<resource-ref>` and `<resource-ref-env>` elements are for standard resource reference and resource reference environment settings. The `<trans-attribute>` element, under `<assembly-descriptor>`, specifies whether transactions are required. (This is discussed in the next section, "Understanding Configuration for Use of Transactions in Message Delivery".)
- See "Configuring the MDB" on page 6-9 for considerations in associating a resource adapter with an MDB during deployment.

Understanding Configuration for Use of Transactions in Message Delivery

In addition to what is discussed in the preceding section, configuration in the `ejb-jar.xml` file specifies whether an MDB uses transactions, and therefore whether message delivery will be transacted. Message delivery is transacted under the following circumstances:

- The `<transaction-type>` subelement of `<message-driven>` in `ejb-jar.xml` has a value of `Container`, and the `<trans-attribute>` subelement of `<container-transaction>` (under the `<assembly-descriptor>` element) has a value of `Required`. This is shown in the example in the preceding section, "Understanding the MDB Configuration for Deployment".

  In this circumstance, if there is an imported transaction, then message delivery and related work are performed within that transaction. If there is no imported transaction, OC4J creates a transaction, and message delivery and related work are performed within that transaction.

- The `<transaction-type>` subelement of `<message-driven>` in `ejb-jar.xml` has a value of `Bean`.

  In this circumstance, the MDB manages the transaction. If a transaction is imported, OC4J will suspend it before the message delivery method call to the MDB, in order to avoid conflict.

Message delivery is *not* transacted if the `<transaction-type>` subelement of `<message-driven>` in `ejb-jar.xml` has a value of `Container`, but the `<trans-attribute>` element has a value of `NotSupported`. If there is an imported transaction in this circumstance, OC4J will suspend the transaction before the message delivery method call to the MDB.

Details of how transactions are started and completed are described in "Message Delivery Semantics" on page 6-12.

Configuring the MDB

After deployment, you can use Application Server Control to change the values of any activation configuration JavaBean properties for an MDB. The new settings take effect when the MDB application is restarted.

In the Application Server Control Console, do the following as appropriate:

1. In the Applications tab of the OC4J Home page, choose the desired application.
2. In the resulting Application Home page, choose the desired EJB module.
3. In the EJBs tab of the resulting EJB Module page, under Message Driven Beans, choose the appropriate MDB.
4. In the resulting Message Driven Bean page, under Message Properties, view the following:
   - Whether the MDB is currently associated with a resource adapter as a listener (`True` or `False`)
   - The name of the resource adapter (if one is used as a listener)
   - A list of activation configuration properties

   The list of activation configuration properties (relevant only when a resource adapter is used as a listener) shows the assembled values, according to settings in the `ejb-jar.xml` file during deployment, and any modified deployed values.
5. In the Message Driven Bean page, use the "Configure Message Listener Properties" feature to access the Configure Message Listener Properties page and specify new values for any of the activation configuration properties, as desired.
6. Apply any changes.

Any new activation configuration property values are shown in the Message Driven Bean page and reflected in the `orion-ejb-jar.xml` file, as shown in Table 6–1. For more information, see the context-sensitive topics "Message Driven Bean Page" and "Configure Message Listener Properties Page" in the Application Server Control online help.

Table 6–1 MDB Activation Configuration Properties

Application Server Control Property	Corresponding XML Entity	Description
Resource Adapter Name	`resource-adapter` attribute of `<message-driven-deployment>` element	The resource adapter to use as a message listener for this MDB (read only). It must support the same message listener type.
Appropriate Name and Deployed Value entries in the list of activation configuration properties	A `<config-property>` element with `<config-property-name>` and `<config-property-value>` subelements	Under the appropriate `<message-driven-deployment>` element, there is a `<config-property>` element for each activation configuration property you updated, with a `<config-property-name>` subelement that indicates the name of the property, and a `<config-property-value>` subelement reflecting your new setting.

Error Conditions for Deployment and Configuration

During resource adapter and MDB deployment and configuration, OC4J verifies settings as follows and throws exceptions as necessary:

- If a resource adapter supports multiple message listener types, OC4J checks that each `<messagelistener-type>` setting in `ra.xml` is unique.
- For each association between an MDB and a resource adapter, OC4J checks that the resource adapter supports the message listener type that the MDB implements.
- For message listener activation, OC4J checks that the activation configuration JavaBean (as specified in the `<activationspec-class>` element) implements the `javax.resource.spi.ActivationSpec` interface, is a JavaBean, and is serializable. OC4J also checks that all configuration properties listed as required in `ra.xml` (under `<required-config-property>` elements) are actually supported by the activation configuration JavaBean .

Understanding the Message Listening Lifecycle

This section discusses the key phases and functionality of the lifecycle for message-listening, covering the following topics:

- MDB Activation
- Message Delivery
- MDB Deactivation

MDB Activation

To enable message delivery from a resource adapter to an MDB (message endpoint), the MDB must be activated whenever it is deployed or started. When an MDB application is deployed, OC4J handles the activation process, as well as automatic reactivation after a system shutdown or failure.

Essentially, activation consists of telling the resource adapter what set of messages the MDB should receive. Message delivery can begin once this information is provided. How the set of messages is defined depends on the type of message listener, such as definitions based on destination type and message selectors for JMS.

Specifically, the following occurs during activation:

- OC4J instantiates the appropriate activation configuration JavaBean, and associates the appropriate resource adapter with it. Property values for the JavaBean are taken from configuration in `ejb-jar.xml`, or according to any subsequent changes made through Application Server Control.
- OC4J calls an activation method on the appropriate resource adapter (specifically, on an instance of a class that implements the SPI `ResourceAdapter` interface) to activate the MDB.
- In the method call to activate the MDB, OC4J passes in the following:
  - The activation configuration JavaBean instance
  - A factory object to use in delivering messages to the MDB

The resource adapter, through the property values of the activation configuration JavaBean, knows what type of messages are expected, where to get them, and how to filter them.

The factory object is an instance of a class that implements the `javax.resource.spi.endpoint.MessageEndpointFactory` interface and is provided by OC4J. See the next section, "Message Delivery", for how the message endpoint factory object is used.

> **Note:** OC4J, in conjunction with the resource adapter, uses a method of the activation configuration JavaBean to check the validity of the bean property settings. If any invalid settings were specified, either in the `ejb-jar.xml` file during deployment or through the Application Server Control Console during subsequent configuration, a failure will result. Refer to the OC4J log files and set an appropriate new value for the parameter in question.

Message Delivery

This section discusses the following aspects of message delivery functionality:

- Message Endpoint Proxy Objects
- Message Delivery Semantics

Message Endpoint Proxy Objects

During message delivery, the resource adapter calls a method on the message endpoint factory it received during MDB activation, to retrieve proxy objects that are used to deliver messages. These proxy objects are instances of a class provided by OC4J that implements the following:

- The `javax.resource.spi.endpoint.MessageEndpoint` interface
- The message listener interface that the MDB implements

Because proxy objects implement the same listener interface as the MDB, the resource adapter can use custom methods of the listener interface to deliver messages.

As discussed in "Understanding Configuration for Use of Transactions in Message Delivery" on page 6-8, message delivery can be either transacted (that is, messages are delivered within transactions) or non-transacted. If delivery is transacted, and the resource adapter passed an `XAResource` object when it retrieved the message endpoint proxy object, then the resource adapter will receive transaction notifications through the `XAResource` instance.

Use of the proxy objects between the resource adapter and the actual message endpoint (MDB) is necessary in case OC4J has to intercept message delivery for any reason, such as to inject a transaction or perform checks.

Notes:

- The message endpoint factory may choose to pool message endpoint proxy objects.
- The resource adapter calls a release method on a message endpoint proxy object when the object is no longer needed, restoring the proxy object into the pool if pooling is used.
- The message endpoint factory provides a utility method, `isDeliveryTransacted()`, for the resource adapter to use in determining if message deliveries will be transacted or not. This information helps the resource adapter to make runtime decisions or perform optimizations.

Message Delivery Semantics

When a resource adapter calls a method for message delivery (the method being specific to the message listener being used), the following must occur:

- The message must be delivered to the actual MDB instance, with any response being returned to the resource adapter.
- Exceptions during message delivery must be handled by OC4J and propagated to the resource adapter, as discussed in "Exceptions from MDB Listener Methods" on page 6-15.
- Transaction semantics must be enforced according to the *J2EE Connector Architecture Specification*, as described in the text that follows.

The following scenarios are considered. (See "Understanding Configuration for Use of Transactions in Message Delivery" on page 6-8 for related information about transaction settings.)

- Container-managed transaction (CMT) demarcation with a setting of `Required` in the MDB configuration, without imported transactions
- Container-managed transaction demarcation with a setting of `NotSupported` in the MDB configuration, without imported transactions
- Container-managed transaction demarcation with a setting of `Required`, with imported transactions
- Container-managed transaction demarcation with a setting of `NotSupported`, with imported transactions
- Bean-managed transaction (BMT) demarcation, without imported transactions
- Bean-managed transaction demarcation, with imported transactions

Furthermore, a resource adapter has the option of exerting control over transaction demarcation by using the `beforeDelivery()` and `afterDelivery()` methods (specified in the `MessageEndpoint` interface) of the message endpoint proxy objects. It can call `beforeDelivery()` before calling the message delivery method on the MDB, and call `afterDelivery()` afterward. We will refer to this as the "before/after" delivery option, and consider it for each of the preceding scenarios.

Independent of this, the resource adapter may choose to pass in an `XAResource` object, to be used for transaction notifications, whenever it retrieves a message endpoint proxy object.

Notes:

- For message delivery that is not transacted, the resource adapter considers a successful return of the message delivery method call to be confirmation of delivery. Any exception from the message endpoint proxy object is interpreted to indicate delivery failure.
- For message delivery that is transacted, a commit of the transaction is interpreted as confirmation of delivery. A rollback is interpreted to indicate delivery failure.

CMT with Required Setting, without Imported Transactions If the resource adapter does not use the before/after delivery option:

- OC4J starts a new transaction before each message delivery method call.
- If the resource adapter provided an `XAResource` object, all transaction notifications are sent to that object.
- OC4J completes the transaction after each message delivery method call.

If the resource adapter uses the before/after delivery option:

- OC4J starts the transaction when `beforeDelivery()` is called.
- If the resource adapter provided an `XAResource` object, all transaction notifications are sent to that object.
- OC4J completes the transaction when `afterDelivery()` is called.

CMT with NotSupported Setting, without Imported Transactions Regardless of whether the resource adapter uses the before/after delivery option:

- OC4J makes no attempt to start a transaction before the message delivery method call or when `beforeDelivery()` is called (as applicable).
- If the resource adapter provided an `XAResource` object, it is ignored.
- OC4J makes no attempt to complete any transaction after the message delivery method call or when `afterDelivery()` is called (as applicable).

CMT with Required Setting, with Imported Transactions Regardless of whether the resource adapter uses the before/after delivery option:

- OC4J does nothing regarding transactions before each message delivery method call or when `beforeDelivery()` is called (as applicable), because an imported transaction already exists.
- If the resource adapter provided an `XAResource` object, all transaction notifications (from the beginning of message delivery or the `beforeDelivery()` call, as applicable) are sent to that object.
- All work done by the MDB is included in the imported transaction.
- OC4J makes no attempt to complete the imported transaction after each message delivery method call or when `afterDelivery()` is called (as applicable), because completion is triggered by the external transaction manager through the resource adapter. OC4J may, however, mark the transaction for rollback based on exceptions that occurred during the call.

CMT with NotSupported Setting, with Imported Transactions If the resource adapter does not use the before/after delivery option:

- OC4J suspends the imported transaction before each message delivery method call.
- If the resource adapter provided an `XAResource` object, it is ignored.
- OC4J resumes the imported transaction after each message delivery method call.

If the resource adapter uses the before/after delivery option:

- OC4J suspends the imported transaction when `beforeDelivery()` is called.
- If the resource adapter provided an `XAResource` object, it is ignored.
- OC4J resumes the imported transaction when `afterDelivery()` is called.

BMT, without Imported Transactions Regardless of whether the resource adapter uses the before/after delivery option:

- OC4J makes no attempt to start, complete, suspend, or resume any transaction.
- If the resource adapter provided an `XAResource` object, it is ignored.

BMT, with Imported Transactions If the resource adapter does not use the before/after delivery option:

- OC4J suspends the imported transaction before each message delivery method call.
- If the resource adapter provided an `XAResource` object, it is ignored.
- OC4J resumes the transaction after each message delivery method call.

If the resource adapter uses the before/after delivery option:

- OC4J suspends the imported transaction when `beforeDelivery()` is called.
- If the resource adapter provided an `XAResource` object, it is ignored.
- OC4J resumes the imported transaction when `afterDelivery()` is called.

MDB Deactivation

Endpoint deactivation is required whenever a resource adapter should stop delivering messages to an MDB, such as when the MDB application that is associated with the resource adapter is undeployed or stopped. Deactivation is handled by OC4J.

Specifically, OC4J calls a deactivation method on the appropriate resource adapter (an instance of a class that implements the SPI `ResourceAdapter` interface) to deactivate the MDB. In the deactivation method call, OC4J passes in the same activation configuration JavaBean instance and message endpoint factory instance that were used in activating the MDB.

Special Conditions in Message Delivery

"Message Delivery" on page 6-11 covered basic functionality. This section describes special conditions, particularly error conditions, during message delivery:

Concurrent Message Delivery

During message delivery, after a resource adapter has retrieved one or more message endpoint proxy objects, the resource adapter has the option of using a single proxy object to serially deliver messages, or multiple proxy objects to deliver messages using different threads. This latter mode is known as *concurrent delivery*.

Note that the message delivery method call and the `beforeDelivery()` and `afterDelivery()` calls (if applicable) are all considered to be part of a single unit of message delivery and all must be called from a single thread.

OC4J supports concurrent delivery as long as a separate proxy object is used in each thread; however, it may attempt to limit the number of message endpoint proxy objects in simultaneous use by throwing `UnavailableException` errors during endpoint proxy creation.

OC4J does *not* support any attempt to concurrently deliver messages using the same endpoint proxy object in different threads. If work is started using a particular proxy object in one thread, the work cannot be handed off to another thread. An attempt to use the same proxy object in different threads results in an exception.

Note: For concurrent message delivery, OC4J maintains a pool of MDBs. Then `min-instances` and `max-instances` attributes in the `ejb-jar.xml` configuration file control the size of the pool.

Exceptions from MDB Listener Methods

This section describes OC4J action in handling exceptions from the message listener method of an MDB, considering the following scenarios:

- Container-managed transaction demarcation with the MDB method running in a transaction context (the case for an MDB `Required` setting)
- Container-managed transaction demarcation with the MDB method running in an unspecified transaction context (the case for an MDB `NotSupported` setting)
- Bean-managed transaction demarcation

CMT with MDB Required Setting For an application exception (`AppException`), OC4J generally attempts to commit the transaction and throw the exception to the resource adapter. However, if the MDB has specified that the only possible outcome of the transaction is a rollback (through the `setRollbackOnly()` method of a `UserTransaction` object on OC4J), then the transaction is rolled back before the exception is thrown to the resource adapter.

For a system exception, OC4J does the following:

- Logs the error.
- Rolls back the transaction.
- Discards the MDB instance (meaning no further methods will be invoked on that instance).
- Throws the exception to the resource adapter after wrapping it in an EJB exception (`EJBException`).

CMT with MDB NotSupported Setting For an application exception, OC4J throws the exception to the resource adapter.

For a system exception, OC4J does the following:

- Logs the error.
- Discards the MDB instance.
- Throws the exception to the resource adapter after wrapping it in an EJB exception.

BMT For an application exception, OC4J throws the exception to the resource adapter.

For a system exception, OC4J does the following:

- Logs the error.
- Marks the transaction for rollback if it was started but not yet completed.
- Discards the MDB instance.
- Throws the exception to the resource adapter after wrapping it in an EJB exception.

Failure During Transacted Message Delivery

This section describes recovery from failure during transacted message delivery, both for transactions managed by OC4J and for imported transactions.

Message Redelivery for Transactions Managed by OC4J

In case of OC4J system failure during transacted message delivery, where the transaction was started by OC4J, there is functionality to notify message providers about the outcomes of all deliveries that were in progress during the failure and are in doubt (prepared but not committed). Recovery is initiated by OC4J. It is left to the message providers to decide if they should attempt redelivery.

Once OC4J is running again, it executes the following recovery steps:

- Restarts the resource adapter instances that were running.
- Obtains an array of `XAResource` objects from each resource adapter, with each object representing a resource manager. In obtaining this array, OC4J passes an array of activation configuration JavaBean instances to each resource adapter, where each activation configuration JavaBean corresponds to an MDB application that was running at the time of system failure.
- Processes the array of `XAResource` objects to create a subset where each object represents a unique resource manager. (Because multiple resource adapters may use the same resource manager, the initial array of `XAResource` objects may have multiple objects corresponding to the same resource manager.)
- Calls methods on each `XAResource` object to query each resource manager for the list of transactions that had been prepared but not yet committed, then completes each such transaction with a commit or rollback call, as appropriate.

Message Redelivery for Imported Transactions

The J2CA specification considers the scenarios that follow for recovery processing from failure during an imported transaction. Recovery is initiated by the EISs through their resource adapters. Note that for imported transactions, OC4J can recover a transaction; however, because it is not the transaction coordinator, it cannot complete the transaction without instruction from the transaction initiator.

An EIS, through its resource adapter, uses the OC4J implementation of the SPI `XATerminator` interface for transaction completion and recovery. This interface includes the methods `prepare()`, `commit()`, `rollback()`, and `recover()`.

For the following discussion, an "active" transaction is one that was not yet prepared, and an "in doubt" transaction is one that was successfully prepared but not yet committed.

If OC4J failure is discovered by the EIS while the transaction is active: The transactional work is presumed aborted. The EIS makes no attempt to wait for OC4J or to attempt recovery processing.

If OC4J failure is discovered by the EIS while the transaction is in doubt: The EIS attempts to complete the transaction. To accomplish this, it attempts to reestablish network connectivity with OC4J until it succeeds. When OC4J recovers, it determines the state of the transaction and completes it as appropriate.

The following specific steps, in accordance with the J2CA specification, are executed in this scenario:

- When OC4J recovers, it restarts the resource adapter that imported the transaction.
- The resource adapter obtains an `XATerminator` object from OC4J.
- The resource adapter re-establishes communication with the EIS.
- The EIS, acting as the transaction coordinator, instructs the resource adapter regarding whether the transaction should be committed or rolled back, and provides the transaction ID through a `javax.transaction.xa.Xid` instance.
- The resource adapter notifies OC4J of the decision to commit or roll back the transaction by calling the `commit()` or `rollback()` method of the `XATerminator` object, passing in the transaction ID.
- OC4J commits or rolls back, as appropriate, all work done on behalf of the transaction.

If EIS failure is discovered by the resource adapter while the transaction is active: The resource adapter aborts it.

If EIS failure is discovered by the resource adapter while the transaction is in doubt: The resource adapter waits for the EIS to recover. When the EIS recovers, it re-establishes network connectivity with the resource adapter and completes the transaction.

The following specific steps, in accordance with the J2CA specification, are executed in this scenario:

- When the EIS recovers, OC4J re-establishes communication.
- The EIS instructs the resource adapter to obtain a list of in-doubt transactions that were imported from the EIS into OC4J.
- The resource adapter obtains a list of transaction IDs (`Xid` instances) of in-doubt transactions. It accomplishes this by calling the `recover()` method of an `XATerminator` object previously obtained.
- The resource adapter forwards the list of transaction IDs to the EIS.
- The EIS, acting as the transaction coordinator, decides for each transaction whether the transaction should be committed or rolled back, and informs the resource adapter of this decision.

- For each transaction, the resource adapter calls the `XATerminator commit()` or `rollback()` method, as appropriate, passing the transaction ID.
- OC4J commits or rolls back, as appropriate, all work performed on behalf of the transaction.

7

Managing Administered Objects

This chapter discusses how to configure and manage resource adapter administered objects, including the following topics:

- Introduction to Administered Objects
- Understanding Deployment Configuration of Administered Objects
- Binding and Editing Administered Objects in OC4J
- Looking Up Administered Objects

Introduction to Administered Objects

This section provides an overview of administered objects used with resource adapters, covering the following:

- What Is an Administered Object?
- Example: What Is an Interaction Spec?
- Example: What Are JMS Topics and Queues?

What Is an Administered Object?

An *administered object* is a utility Java object, provided with a resource adapter, for use by an application component in conjunction with that resource adapter. A resource adapter may or may not provide administered objects. Typically, an administered object is used for messaging and is specific to a particular messaging style or message provider. For example, a Java Message System (JMS) "queue" object may be required to synchronously send and receive messages to Oracle Advanced Queueing through the JMS API. An administered object in Oracle J2CA acts as a placeholder for an entity defined elsewhere, such as for the queue object that is defined in the JMS environment. (Administered objects are not necessary for asynchronous messaging, the setup of which involves activation spec JavaBeans instead, as described in "Understanding the Resource Adapter Configuration" on page 6-4.)

Classes for administered objects are supplied by resource adapter providers. Each class must be a JavaBean class implementing some appropriate standard interface. Configuration of an administered object includes the following:

- Setting any properties, which is accomplished by the resource adapter provider through the `ra.xml` file for deployment, or later by the OC4J administrator through Application Server Control
- Binding the object into JNDI for lookup by application components, which you accomplish through Application Server Control

The next couple of sections discuss examples of two particular kinds of administered object. Many more examples are possible.

Example: What Is an Interaction Spec?

The CCI `InteractionSpec` interface provides the mechanism for holding properties for interactions. (See "What Are the Interface Libraries of the J2EE Connector Architecture?" on page 1-8 for information about interactions and the CCI `Interaction` interface.) An interaction spec implementation must support standard properties that include an EIS function name and "interaction verb", which specifies whether an interaction execution results in synchronously receiving a data record, synchronously sending a data record, or both. There are also standard properties to provide information to an interaction object about result set requirements (fetch size, for example).

An interaction spec instance can optionally be registered as an administered object, allowing application components to look it up in the JNDI namespace through a standard resource environment reference that was specified during deployment.

See version 1.5 of the *J2EE Connector Architecture Specification* for additional information.

Example: What Are JMS Topics and Queues?

JMS includes two modes of operation, known as "publish/subscribe" and "point-to-point". The publish/subscribe mode uses "topic" objects defined in JMS, and the point-to-point mode uses "queue" objects defined in JMS. In the J2EE Connector Architecture, these messaging modes involve the use of administered objects to act as placeholders for topics and queues, respectively. These administered objects, created by an administrator for use by a JMS client, provide a convenient way to specify configuration information for JMS.

Point-to-point messaging uses queues of messages, with a client sending messages to a single queue and typically having all its messages delivered from a single queue. Most queues are static resources created by an administrator. JMS defines how a client sends messages to a queue and receives messages from a queue.

Publish/subscribe messaging uses message topics, where each topic represents a node of a hierarchy of possible message subjects. The publish/subscribe model specifies how JMS clients can publish messages to a topic or subscribe to messages from a topic. In JMS, a topic object more specifically acts as a broker, gathering messages published to it and distributing messages to subscribers as appropriate. Publishers and subscribers remain independent of each other, and the topic entity adjusts automatically as publishers and subscribers are added or dropped.

A topic is represented by an instance of a class that implements the `javax.jms.Topic` interface. A queue is represented by an instance of a class that implements the `javax.jms.Queue` interface. Each encapsulates a provider-specific message destination address. More specifically, a `Topic` object encapsulates a provider-specific topic name, while a `Queue` object encapsulates a provider-specific queue name. As administered objects, topics and queues can contain provider-specific configuration information as well.

The `Topic` and `Queue` interfaces do not depend on JNDI, but it is a JMS convention to use JNDI to bind and look up `Topic` and `Queue` instances. Topics and queues are used for examples later in this chapter.

Note: Topics and queues support concurrent use.

Understanding Deployment Configuration of Administered Objects

Administered objects are configured for deployment in the `ra.xml` file, typically by the resource adapter provider, as in the following example for JMS topics and queues. The `<adminobject>` element is a subelement of the `<resourceadapter>` element.

```
<connector ... >
   ...
   <resourceadapter>
      ...
      <!--   Queue admin object  -->
      <adminobject>
         <adminobject-interface>javax.jms.Queue</adminobject-interface>
         <adminobject-class>
            oracle.j2ee.ra.jms.generic.AdminObjectQueueImpl
         </adminobject-class>
         <config-property>
            <config-property-name>jndiName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>MQQ</config-property-value>
         </config-property>
         <config-property>
            <config-property-name>resourceProviderName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>MQSeries</config-property-value>
          </config-property>
      </adminobject>

      <!--   Topic admin object  -->
      <adminobject>
         <adminobject-interface>javax.jms.Topic</adminobject-interface>
         <adminobject-class>
            oracle.j2ee.ra.jms.generic.AdminObjectTopicImpl
         </adminobject-class>
         <config-property>
            <config-property-name>jndiName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>MQT</config-property-value>
         </config-property>
         <config-property>
            <config-property-name>resourceProviderName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>MQSeries</config-property-value>
         </config-property>
      </adminobject>
      ...
   </resourceadapter>
   ...
</connector>
```

The `<adminobject-class>` element specifies the administered object class, and the `<adminobject-interface>` element specifies the interface that this class implements.

The `<config-property>` elements and their subelements specify the name, type, and deployed value of properties of the administered objects.

For queues, in this example, the JMS `Queue` interface is implemented by the class `AdminObjectQueueImpl`. This administered object has the following properties:

- `jndiName`: type `String`, deployed value `MQQ`
- `resourceProviderName`: type `String`, deployed value `MQSeries`

For topics, in this example, the JMS `Topic` interface is implemented by the class `AdminObjectTopicImpl`. This administered object has the following properties:

- `jndiName`: type `String`, deployed value `MQT`
- `resourceProviderName`: type `String`, deployed value `MQSeries`

(You can change the deployed values later, through Application Server Control, as described in the next section, "Binding and Editing Administered Objects in OC4J".)

Note: OC4J instantiates each administered object during resource adapter deployment, as well as at OC4J startup or restart, using the configured property values.

Binding and Editing Administered Objects in OC4J

To use an administered object in OC4J, you must initially use the Application Server Control "Create" function to bind it to a JNDI location. You can also edit property values at this time, or at a later time. Changes take effect when the resource adapter is restarted. The following sections describe the steps involved:

- Create and Bind an Administered Object
- View or Edit an Administered Object

OC4J binds each administered object into the JNDI namespace. For resources to be directly visible from `java:comp/env`, there must be an appropriate resource reference. You would accomplish this using a `<resource-ref>` element in either the `web.xml` file (for Web modules) or the `ejb-jar.xml` file (for EJB modules). Even without such a resource reference, though, you can do a lookup according to the JNDI location that you specify when you bind the administered object.

Create and Bind an Administered Object

To use an administered object you must create it and bind it, using the following steps in the Application Server Control Console:

1. Go to the Administered Objects tab from the appropriate Resource Adapter Home page.
2. Choose the "Create" function.
3. In the resulting Create Administered Object page, choose an administered object class. The list of available classes is according to the `<adminobject-class>` elements in the `ra.xml` file.
4. In the next Create Administered Object page, specify a JNDI location.
5. Optionally specify any new values for the configuration properties. The list of available configuration properties and their initial values is according to the `ra.xml` configuration of the administered object class you select.
6. Apply the changes.

Also see the context-sensitive topics "Create Administered Object - Select Class Page" and "Create Administered Object Page" in the Application Server Control online help.

When you create and bind an administered object, OC4J makes appropriate entries in the `oc4j-connectors.xml` file. For each administered object, OC4J writes an `<adminobject-config>` element, which is a subelement of the top-level `<connector>` element. The `<adminobject-config>` element includes the following:

- `location` attribute for the JNDI location, such as:

```
<adminobject-config location="mqjms/MyQ" >
```

- `<adminobject-class>` subelement for the name of the implementing class, such as:

```
<adminobject-class>
   oracle.j2ee.ra.jms.generic.AdminObjectQueueImpl
</adminobject-class>
```

- `<config-property>` subelement for each property of the administered object, with `name` and `value` attributes, such as:

```
<config-property name="jndiName" value="MQQ" />
<config-property name="resourceProviderName" value="MQSeries" />
```

Table 7–1 summarizes the correlation between Application Server Control properties and corresponding attributes and subelements of `<adminobject-config>` elements in the `oc4j-connectors.xml` file.

Table 7–1 Administered Object JNDI Location and Configuration Properties

Application Server Control Property	Corresponding XML Entity	Description
Object Class	`<adminobject-class>` subelement	JavaBean class that represents an administered object.
JNDI Location	`location` attribute	JNDI location of an administered object instance.
Name (under configuration properties)	`name` attribute of `<config-property>` subelement	Name of a configuration property of an administered object.
Value (under configuration properties)	`value` attribute of `<config-property>` subelement	Value of a configuration property of an administered object.

Note: You can create and bind multiple administered objects for any given administered object type that is defined in the `ra.xml` file.

View or Edit an Administered Object

You can edit the property values of an administered object you previously created. Existing administered objects are identified by their JNDI locations (and you cannot change a location).

Use the following steps in the Application Server Control Console:

1. Go to the Administered Objects tab of the appropriate Resource Adapter Home page.

2. Choose the JNDI location of the administered object you want to edit.
3. In the resulting Administered Object page, you can view the configuration properties of the administered object. Additionally, you can specify new values for any editable configuration properties as desired.
4. Apply the changes.

Also see the context-sensitive topic "Administered Object Page" in the Application Server Control online help.

Looking Up Administered Objects

Once administered objects have been deployed and configured, as described previously, they can be accessed by application components through JNDI lookups.

For this discussion, following up on earlier discussion of JMS topics and queues in this chapter, assume administered objects for topics and queues have been deployed with JNDI locations `mqjms/MyT` and `mqjms/MyQ`, respectively.

Further assume the topics and queues will be used in a servlet, with the following resource environment reference mapping in the `web.xml` file:

```
<web-app ... >
   ...
   <resource-env-ref>
      <resource-env-ref-name>jms/TOPIC1</resource-env-ref-name>
      <resource-env-ref-type>javax.jms.Topic</resource-env-ref-type>
   </resource-env-ref>
   <resource-env-ref>
      <resource-env-ref-name>jms/QUEUE1</resource-env-ref-name>
      <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
   </resource-env-ref>
   ...
</web-app>
```

And assume there has been Web module configuration, such as through Application Server Control, that results in the following mapping, as reflected in the `orion-web.xml` file:

```
<orion-web-app ...>
   ...
      <resource-env-ref-mapping name="jms/QUEUE1" location="mqjms/MyQ" />
      <resource-env-ref-mapping name="jms/TOPIC1" location="mqjms/MyT" />
   ...
</orion-web-app>
```

The preceding entries map actual JNDI locations (such as `mqjms/MyQ`) to logical JNDI locations (such as `jms/QUEUE1`). These entries allow a servlet to look up a topic as follows:

```
InitialContext ic = new InitialContext();
Topic ourT = (Topic)ic.lookup("java:comp/env/jms/TOPIC1");
```

And allow a servlet to look up a queue as follows:

```
InitialContext ic = new InitialContext();
Queue ourQ = (Queue)ic.lookup("java:comp/env/jms/QUEUE1");
```

A

OC4J Resource Adapter Configuration Files

This appendix begins with an overview of resource adapter configuration files, then provides detailed reference information about the OC4J-specific files. The following topics are covered:

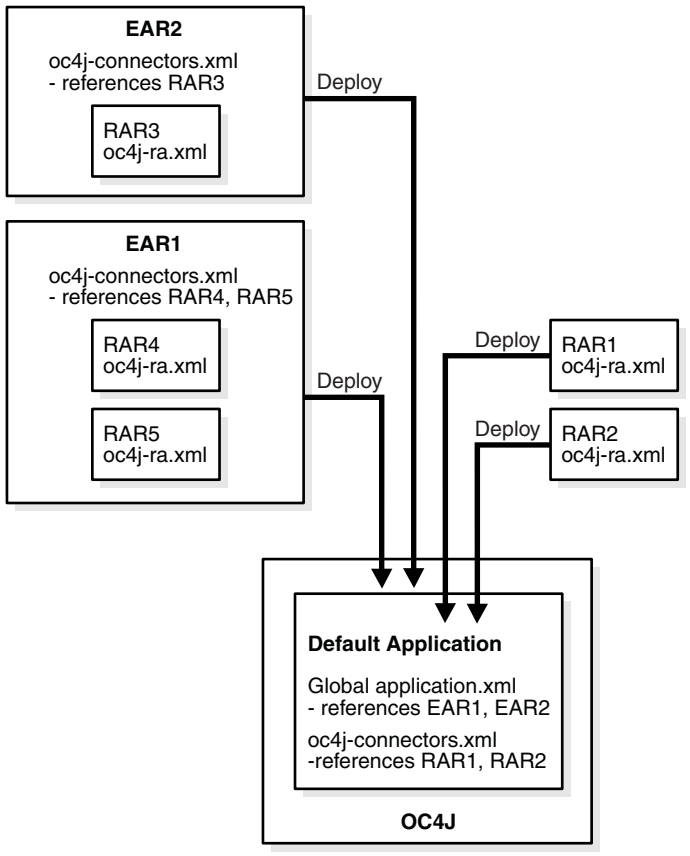
Overview of Resource Adapter Configuration Files

This section provides an overview of the resource adapter configuration files: the standard `ra.xml` file and the Oracle `oc4j-ra.xml` and `oc4j-connectors.xml` files.

Figure A–1 shows the relationship between OC4J XML files and EAR and RAR files for deployment, as follows:

- An `oc4j-ra.xml` file is associated with each RAR file, whether for a standalone or in-application resource adapter, to set configuration for the resource adapter being deployed.
- An `oc4j-connectors.xml` file is associated with each EAR file (for applications using resource adapters), to reference the resource adapters being deployed with the application as well as set additional configuration.
- Also, an `oc4j-connectors.xml` file is associated with the OC4J default application, to reference standalone resource adapters as well as set additional configuration.

Figure A–1 Deployment of OC4J-Specific XML Configuration Files for Oracle J2CA



Standard ra.xml Configuration File

An `ra.xml` file for a resource adapter is supplied by the resource adapter provider and is included in the RAR file for deployment. When you subsequently configure a resource adapter, as discussed throughout this manual, entries in `ra.xml` typically serve as defaults, which you can override through Application Server Control. Any configuration settings that override `ra.xml` entries are reflected in corresponding `oc4j-ra.xml` entries generated by OC4J.

Here are portions of a sample J2CA 1.5 `ra.xml` file:

```
<outbound-resourceadapter>
   <connection-definition>
      <managedconnectionfactory-class>
         com.example.ManagedConnectionFactoryImpl
      </managedconnectionfactory-class>
      <connectionfactory-interface>
         javax.resource.cci.ConnectionFactory
      </connectionfactory-interface>
      <connectionfactory-impl-class>
         com.example.ConnectionFactoryWrapper
      </connectionfactory-impl-class>
      <connection-interface>javax.resource.cci.Connection</connection-interface>
      <connection-impl-class>
         com.example.ConnectionWrapper
      </connection-impl-class>
      <config-property>
```

```
            <config-property-name>ServerName</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>www.example.com</config-property-value>
         </config-property>
         <config-property>
            <config-property-name>ConnectionUrl</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
            <config-property-value>
               jdbc:oracle:thin:@www.example.com:1521/myservice
            </config-property-value>
         </config-property>
      </connection-definition>
...
</outbound-resourceadapter>
```

Refer to the J2CA specification for detailed information about `ra.xml` and the elements and attributes it supports.

Oracle oc4j-ra.xml Configuration File

The `oc4j-ra.xml` file contains OC4J-specific configuration for a resource adapter. When you deploy a RAR file that does not contain an `oc4j-ra.xml` file, OC4J will create one automatically, using settings from the `ra.xml` file as default values as applicable. Then, whenever you use Application Server Control Console to create or edit a connection factory, OC4J updates the `oc4j-ra.xml` file.

This file declares connection factories, with each `<connector-factory>` element mapping to a `<connection-definition>` element in `ra.xml`. This is a many-to-one relationship, as there can be multiple `<connector-factory>` elements corresponding to a single `<connection-definition>` element. The `<connector-factory>` elements are all under the root `<oc4j-connector-factories>` element.

Subelements of `<connector-factory>` include `<connection-pooling>`, to set up connection pooling for the factory, and `<security-config>`, to set up container-managed sign-on. Each connector factory can have configuration for a private connection pool, or can use a shared connection pool that is set up through a `<connection-pool>` subelement of `<oc4j-connector-factories>`.

You can optionally package an `oc4j-ra.xml` file with the RAR file for deployment. It is typical for a resource adapter provider to supply a number of configuration files specific to particular application servers, in addition to their `ra.xml` file, and this may include an `oc4j-ra.xml` file. When you deploy a RAR file that includes `oc4j-ra.xml`, OC4J does not generate an `oc4j-ra.xml` file or alter your packaged `oc4j-ra.xml` file during deployment. It simply accepts the settings in the packaged file and exposes them as assembled values through Application Server Control, which of course you can later edit if desired.

See "What Are the Packaging and Deployment Features?" on page 1-9 for related information.

Oracle oc4j-connectors.xml Configuration File

For each EAR file deployed to OC4J, the `oc4j-connectors.xml` file lists the resource adapters deployed with the application, using a `<connector>` element for each adapter. Subelements of `<connector>`, for configuration relating to a particular resource adapter, include `<adminobject-config>` to configure administered objects, and `<security-permission>` to set up permissions.

An `oc4j-connectors.xml` file is also used to store the list of standalone resource adapters in OC4J, indicating the number of resource adapters that were deployed outside of EAR files and are to be globally available. This `oc4j-connectors.xml` file is associated with the OC4J default application.

For each application, there is a `<connectors>` element in the `orion-application.xml` file to indicate where the associated `oc4j-connectors.xml` file is located. For standalone resource adapters, `oc4j-connectors.xml` is located by default in the OC4J `j2ee/`*`instance`*`/config` directory, as indicated in the `<connectors>` element in the global OC4J `application.xml` file. In the directory name, *`instance`* is the name of an OC4J instance (`home` by default in an Oracle Application Server environment and always `home` in a standalone environment).

If you do not provide an `oc4j-connectors.xml` file in an EAR file, OC4J creates it for you. The global `oc4j-connectors.xml` file for standalone resource adapters already exists by default.

Hierarchy of oc4j-ra.xml

Here is an overview of the `oc4j-ra.xml` element hierarchy.

```
<oc4j-connector-factories>
   <connection-pool>
      <property>
   <connector-factory>
      <description>
      <connectionfactory-interface>
      <config-property>
      <log>
         <file>
      <connection-pooling>
         <property>
         <use-connection-pool>
      <xa-recovery-config>
         <password-credential>
            <username>
            <password>
      <security-config>
         <principal-mapping-entries>
            <description>
            <principal-mapping-entry>
               <initiating-user>
               <res-user>
               <res-password>
            <default-mapping>
               <res-user>
               <res-password>
         <principal-mapping-interface>
            <impl-class>
            <property>
         <jaas-module>
            <jaas-application-name>
```

Elements and Attributes of oc4j-ra.xml

This section is an alphabetical dictionary of elements of the `oc4j-ra.xml` file. See the preceding section, "Hierarchy of oc4j-ra.xml", if you are interested in the hierarchy.

Note: Where attributes are discussed, note that attribute values are always set inside quotes: `attribute="value"`

<config-property>

Parent element: <connector-factory>

Child elements: None

Required? Optional; zero or more

Each `<config-property>` element specifies the name and value of a property setting for the connection factory. These map to `<config-property>` elements in the corresponding connection definition in the `ra.xml` file.

Table A–1 <config-property> Attributes

Name	Description
name	Values: String Default: n/a (required) The name (from `ra.xml`) of the connection property being set.
value	Values: String Default: n/a (required) The desired value of the connection property being set.

<connection-pool>

Parent element: <oc4j-connector-factories>

Child elements: <property>

Required? Optional; zero or more

This element specifies the name of a shared connection pool, and optionally specifies property settings of the pool through `<property>` subelements.

Note: Do not confuse the `<connection-pool>` element with the `<connection-pooling>` element, which is a subelement of `<connector-factory>`.

Table A–2 <connection-pool> Attributes

Name	Description
name	Values: String Default: n/a (required) The desired name of the shared connection pool.

<connection-pooling>

Parent element: <connector-factory>

Child elements: <property>, <use-connection-pool>

Required? Optional; zero or one

This element specifies a connection pool for use by the connection factory defined through the parent `<connector-factory>` element.

The `<connection-pool>` element either specifies a private connection pool with its own settings, through `<property>` subelements, or uses a shared pool, through a `<use-connection-pool>` subelement.

The absence of this element is equivalent to an attribute setting of `use="none"`, disabling connection pooling.

See "Configuring Connection Pooling in OC4J" on page 3-4 and "Sharing Connection Pools" on page 3-10 for related information about connection pools.

Note: Do not confuse the `<connection-pooling>` element with the `<connection-pool>` element, which is a subelement of `<oc4j-connector-factories>`.

Table A–3 <connection-pooling> Attributes

Name	Description
use	Values: shared | private | none Default: No default Use "`shared`" to use a shared connection pool, "`private`" to specify a private connection pool, or "`none`" to disable connection pooling for the connection factory.

<connectionfactory-interface>

Parent element: <connector-factory>

Child elements: None

Required? Optional; zero or one

The value of this element indicates the fully qualified name of the Java interface upon which connection factories defined through the parent `<connector-factory>` element are based:

```
<connectionfactory-interface>interfacename</connectionfactory-interface>
```

A connection factory object is an instance of a class that implements this interface. This must be one of the interfaces specified for the corresponding connection definition in the `ra.xml` file.

The `<connectionfactory-interface>` element has no attributes.

Important: This element is not required according to the schema definition, for backward compatibility reasons, but its omission will result in an exception.

<connector-factory>

Parent element: <oc4j-connector-factories>

Child elements: <config-property>, <connection-pooling>, <connectionfactory-interface>, <description>, <log>, <security-config>, <xa-recovery-config>

Required? Optional; zero or more

This element, through its attributes and subelements, specifies a connection factory for the resource adapter.

Table A–4 <connector-factory> Attributes

Name	Description
connector-name	Values: String Default: n/a (required) The name of the resource adapter instance. This name can be the same as the connector name specified during deployment for a standalone resource adapter (in the `deploymentName` property of the `deploy` task or through the Application Server Control Console deployment page) or the RAR file name, without the `.rar` extension, for a resource adapter deployed within an EAR file. The connector name or RAR file name (without the extension) corresponds to the `name` attribute of the applicable `<connector>` element in the `oc4j-connectors.xml` file.
location	Values: String Default: n/a (required) A JNDI location to which the connection factory object will be bound.

<default-mapping>

Parent element: <principal-mapping-entries>

Child elements: <res-user>, <res-password>

Required? Optional; zero or one

This element, through its subelements, specifies a default resource principal. If an initiating principal does not match any of those that are mapped to resource principals through `<principal-mapping-entry>` elements, then the default resource principal is used.

The `<default-mapping>` element has no attributes.

<description>

Parent element: <connector-factory>

Child elements: None

Required? Optional; zero or one

The value of this optional element describes the connection factory:

```
<description>Description of connection factory.</description>
```

The <description> element has no attributes.

<description>

Parent element: <principal-mapping-entries>

Child elements: None

Required? Optional; zero or one

The value of this optional element describes the principal mapping entries:

```
<description>Description of principal mapping entries.</description>
```

The <description> element has no attributes.

<file>

Parent element: <log>

Child elements: None

Required? Required within <log>; one only

This element, through its path attribute, specifies the absolute or relative path and name of a log file where OC4J will write logging and tracing messages relating to the resource adapter and EIS.

Table A–5 <file> Attributes

Name	Description
path	Values: String Default: No default The absolute or relative path and name of the log file.

<impl-class>

Parent element: <principal-mapping-interface>

Child elements: None

Required? Required; one only

The value of this element indicates the fully qualified name of the principal mapping class:

```
<impl-class>classname</impl-class>
```

The <impl-class> element has no attributes.

<initiating-user>

Parent element: <principal-mapping-entry>

Child elements: None

Required? Required within <principal-mapping-entry>; one only

The value of this element is the user name of an initiating principal, to be mapped to a resource principal through an associated <res-user> (and <res-password>) element:

```
<principal-mapping-entry>
   <initiating-user>name</initiating-user>
   <res-user>name</res-user>
   <res-password>pwd</res-password>
</principal-mapping-entry>
```

The <initiating-user> element has no attributes.

The OC4J property jaas.username.simple determines whether realm names are prefixed in user names for returned principals. With the default "true" setting, realm names are not prefixed. If you configure and use custom realms, you must explicitly set this property to "false" to ensure that OracleAS JAAS Provider authentication and authorization work properly.

To specify a "false" setting, use a <property> subelement of the <jazn> element (in orion-application.xml for application level, or in the instance-level jazn.xml file for OC4J instance level), as follows:

```
<jazn ... >
...
<property name="jaas.username.simple" value="false" />
...
</jazn>
```

Important: Always set jaas.username.simple to "false" when multiple realms are configured.

See the *Oracle Containers for J2EE Security Guide* for more information about the jaas.username.simple property.

<jaas-application-name>

Parent element: <jaas-module>

Child elements: None

Required? Required within <jaas-module>; one only

The value of this element indicates the name of the JAAS login module used for container-managed sign-on:

```
<jaas-module>
   <jaas-application-name>modulename</jaas-application-name>
</jaas-module>
```

The <jaas-application-name> element has no attributes.

<jaas-module>

Parent element: <security-config>

Child elements: <jaas-application-name>

Required? Optional; zero or one (but required if `use="jaas-module"` in the parent `<security-config>`)

This element is for the "JAAS login module" mechanism for container-managed sign-on. In this mechanism, a developer plugs in a JAAS login module (specified in the `<jaas-application-name>` subelement) under the application to use a particular authentication mechanism. A login module may, for example, simply implement a user/password authentication mechanism, or may use more sophisticated technology such as by connecting to a Smart Card.

Use of the `<jaas-module>` element must be combined with a setting of `use="jaas-module"` in the parent `<security-config>` element (each requires the other).

The `<jaas-module>` element has no attributes.

<log>

Parent element: <connector-factory>

Child elements: <file>

Required? Optional; zero or one

This specifies a log file, through its `<file>` subelement.

See "Configuring OC4J Logging for Connection Factories" on page 3-12 for related information about log files and their configuration.

The `<log>` element has no attributes.

<oc4j-connector-factories>

Parent element: n/a (root)

Child elements: <connection-pool>, <connector-factory>

Required? Required; one only

This is the top-level element of the `oc4j-ra.xml` file, encompassing configuration of connection factories and connection pools for a single resource adapter deployed to OC4J.

Table A–6 <oc4j-connector-factories> Attributes

Name	Description
schema-major-version	Values: String Default: No default The major version number of the `oc4j-ra.xml` XSD. If you create `oc4j-ra.xml` manually, set this attribute to 10 for use with the OC4J 10.1.3 implementation.

Table A–6 (Cont.) <oc4j-connector-factories> Attributes

Name	Description
schema-minor-version	Values: String Default: No default The minor version number of the `oc4j-ra.xml` XSD. If you create `oc4j-ra.xml` manually, set this attribute to 0 for use with the OC4J 10.1.3 implementation.

Note: These attributes do not appear directly in the XSD for `ocj4-ra.xml`. They are according to the `attributeGroup` specification in the top-level OC4J XSD.

<password>

Parent element: <password-credential>

Child elements: None

Required? Optional; zero or one

The value of this element is for a password for the associated user name (through the corresponding `<username>` element), for XA recovery sign-on through the connection factory.

There is a corresponding `<username>` element for the associated user name.

Specify the password setting in one of the following ways:

- Direct password: the value is the password itself
- Indirect Password: the value is a right-arrow followed by a key that OC4J uses for a lookup in the User Manager (such as through the `jazn-data.xml` file)
- No password: empty content

For an indirect password, the key is often just the user name, as in the following example:

```
...
<password-credential>
   <username>jmsuser</username>
   <password>->jmsuser</password>
</password-credential>
...
```

The `<password>` element has no attributes.

<password-credential>

Parent element: <xa-recovery-config>

Child elements: <username>, <password>

Required? Optional; zero or more

This element, through its <username> and <password> subelements, specifies sign-on information for XA recovery.

You can use multiple <password-credential> elements to specify multiple XA recovery sign-on identities.

The <password-credential> element has no attributes.

<principal-mapping-entries>

Parent element: <security-config>

Child elements: <description>, <principal-mapping-entry>, <default-mapping>

Required? Optional; zero or one (but required if use="principal-mapping-entries" in the parent <security-config>)

This element is for the "principal mapping entries" mechanism for container-managed sign-on. In this mechanism, principal mappings are specified through OC4J-specific configuration settings ("declaratively").

Use of the <principal-mapping-entries> element must be combined with a setting of use="principal-mapping-entries" in the parent <security-config> element (each requires the other).

The <principal-mapping-entries> element has no attributes.

<principal-mapping-entry>

Parent element: <principal-mapping-entries>

Child elements: <initiating-user>, <res-user>, <res-password>

Required? Optional; zero or more

For the "principal mapping entries" mechanism, there can be any number of <principal-mapping-entry> elements. Each one specifies, through its subelements, a mapping between an initiating principal and a resource principal for the EIS connection.

The <principal-mapping-entry> element has no attributes.

<principal-mapping-interface>

Parent element: <security-config>

Child elements: <impl-class>, <property>

Required? Optional; zero or one (but required if use="principal-mapping-interface" in the parent <security-config>)

This element is for the "principal mapping class" mechanism for container-managed sign-on. In this mechanism, principal mappings are specified through a *principal mapping class* ("programmatically"). A principal mapping class is one that implements the oracle.j2ee.connector.PrincipalMapping interface, and its name is specified in the <impl-class> subelement. A developer can implement the PrincipalMapping interface directly, or can extend the

`oracle.j2ee.connector.AbstractPrincipalMapping` class that is supplied by Oracle for convenience.

Properties of a principal mapping class are specific to the particular implementation. Property settings for a principal mapping instance are configured through `<property>` subelements.

Use of the `<principal-mapping-interface>` element must be combined with a setting of `use="principal-mapping-interface"` in the parent `<security-config>` element (each requires the other).

The `<principal-mapping-interface>` element has no attributes.

<property>

Parent element: <connection-pool>

Child elements: None

Required? Optional; zero or more

Use a `<property>` element for each property you want to set for the shared connection pool defined in the parent `<connection-pool>` element.

> **Note:** This element has the same usage and attributes as the `<property>` subelement of `<connection-pooling>`, except `initial-capacity` is not supported.

Table A–7 <property> Attributes

Name	Description
name	Values: maxConnections | minConnections | scheme | waitTimeout | inactivity-timeout | inactivity-timeout-check Default: n/a (required) The name of the property being set. See "Configuring Connection Pooling in OC4J" on page 3-4 for descriptions of these properties.
value	Values: String Default: n/a (required) The desired value of the property being set. See "Configuring Connection Pooling in OC4J" on page 3-4 for supported values of these properties.

<property>

Parent element: <connection-pooling>

Child elements: None

Required? Optional; zero or more

Use a `<property>` element for each property you want to set for the private connection pool defined in the parent `<connection-pooling>` element.

Note: This element has the same usage and attributes as the `<property>` subelement of `<connection-pool>`, except this element also supports the `initial-capacity` property.

Table A–8 <property> Attributes

Name	Description
name	Values: maxConnections | minConnections | initial-capacity | scheme | waitTimeout | inactivity-timeout | inactivity-timeout-check Default: n/a (required) The name of the property being set. See "Configuring Connection Pooling in OC4J" on page 3-4 for descriptions of these properties.
value	Values: String Default: n/a (required) The desired value of the property being set. See "Configuring Connection Pooling in OC4J" on page 3-4 for supported values of these properties.

<property>

Parent element: <principal-mapping-interface>

Child elements: None

Required? Optional; zero or more

There can be any number of `<property>` elements to specify property settings for an instance of the principal mapping class, depending on how many properties the class itself has. For example, properties can specify a default user name and password, the path of a principal mapping file, information for an LDAP connection, or a default mapping.

Table A–9 <property> Attributes

Name	Description
name	The name of the property being set (a property of the principal mapping class).
value	The desired value of the property.

<res-password>

Parent element: <default-mapping>

Child elements: None

Required? Required within `<default-mapping>`; one only

The value of this element specifies the password for the default resource principal of the associated `<res-user>` element, either directly or indirectly. This uses the same functionality as discussed for <password> on page A-11.

The <res-password> element has no attributes.

<res-password>

Parent element: <principal-mapping-entry>

Child elements: None

Required? Required within <principal-mapping-entry>; one only

The value of this element specifies the password for the resource principal of the associated <res-user> element, either directly or indirectly. This uses the same functionality as discussed for <password> on page A-11.

The <res-password> element has no attributes.

<res-user>

Parent element: <default-mapping>

Child elements: None

Required? Required within <default-mapping>; one only

The value of this element is the user name of the default resource principal for the principal mapping entries mechanism. There is also an associated <res-password> element:

```
<default-mapping>
   <res-user>name</res-user>
   <res-password>pwd</res-password>
</default-mapping>
```

The <res-user> element has no attributes.

<res-user>

Parent element: <principal-mapping-entry>

Child elements: None

Required? Required within <principal-mapping-entry>; one only

The value of this element is the user name of a resource principal, being mapped from an initiating principal through an associated <initiating-user> element. Also use an associated <res-password> element:

```
<principal-mapping-entry>
   <initiating-user>name</initiating-user>
   <res-user>name</res-user>
   <res-password>pwd</res-password>
</principal-mapping-entry>
```

The <res-user> element has no attributes.

<security-config>

Parent element: <connector-factory>

Child elements: <principal-mapping-entries>, <principal-mapping-interface>, <jaas-module>

Required? Optional; zero or one

This element, through its `use` attribute and appropriate subelements, specifies container-managed sign-on and any associated *principal mapping*. Principal mapping is defined in the J2CA specification and refers to mapping from an initiating principal (such as an OC4J user) to a resource principal (resource user, or EIS user). The `<security-config>` element can specify any of the following:

- Declarative container-managed sign-on through principal mapping entries
- Programmatic container-managed sign-on through an implementation of the principal mapping interface
- Programmatic container-managed sign-on through a JAAS application module
- Disabling of container-managed sign-on

See the *Oracle Containers for J2EE Security Guide* for information about container-managed sign-on versus component-managed sign-on, and for additional information and examples regarding `<security-config>` and its subelements.

Table A–10 <security-config> Attributes

Name	Description
use	Values: none | principal-mapping-entries | principal-mapping-interface | jaas-module Default: No default Use "`none`" to disable container-managed sign-on (such as for component-managed sign-on), or use "`principal-mapping-entries`", "`principal-mapping-interface`", or "`jaas-module`" to indicate the mode of container-managed sign-on. These values for container-managed sign-on reflect the names of the corresponding subelements of `<security-config>`.

<use-connection-pool>

Parent element: <connection-pooling>

Child elements: None

Required? Optional; zero or one (but required if `use="shared"` in parent element, or OC4J will throw an exception)

When the parent `<connection-pooling>` element has the attribute setting `use="shared"` to use a shared connection pool, the value of the `<use-connection-pool>` subelement specifies the name of the shared connection pool to use:

```
<use-connection-pool>connectionpoolname</use-connection-pool>
```

This corresponds to the `name` attribute of the `<connection-pool>` element that specifies the shared pool.

The `<use-connection-pool>` element has no attributes.

<username>

Parent element: <password-credential>

Child elements: None

Required? Optional; zero or one

The value of this element indicates a user name for XA recovery sign-on through the connection factory:

```
...
<password-credential>
   <username>name</username>
   <password>pwd</password>
</password-credential>
...
```

As shown, there is a corresponding `<password>` element for the associated password.

The `<username>` element has no attributes.

<xa-recovery-config>

Parent element: <connector-factory>

Child elements: <password-credential>

Required? Optional; zero or one

This element, through one or more `<password-credential>` subelements, specifies sign-on information for XA recovery for the connection factory. See "Understanding and Configuring Transaction Recovery" on page 4-15 for related information and concepts.

The `<xa-recovery-config>` element has no attributes.

> **Note:** If no sign-on credentials are specified for XA recovery, the first fallback is to use what is specified through the `<security-config>` element. If there is no `<security-config>` element, the next fallback is to sign on according to the OC4J user at the time of recovery execution.

Sample oc4j-ra.xml

This section shows a sample `oc4j-ra.xml` file that configures connection factories for a JMS resource adapter, including setup of connection pooling (both private and shared) and container-managed sign-on.

```
<?xml version="1.0"?>

<oc4j-connector-factories xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation=
"http://xmlns.oracle.com/oracleas/schema/oc4j-connector-factories-10_0.xsd"
schema-major-version="10" schema-minor-version="0" >
  <connector-factory location="OracleASjms/MyXAQCF" connector-name="OracleASjms">
     <config-property name="jndiLocation" value="jms/XAQueueConnectionFactory"/>
     <connection-pooling use="private">
        <property name="waitTimeout" value="300" />
        <property name="scheme" value="fixed_wait" />
        <property name="maxConnections" value="50" />
        <property name="minConnections" value="0" />
     </connection-pooling>
     <security-config use="principal-mapping-entries">
        <principal-mapping-entries>
           <default-mapping>
              <res-user>anonymous</res-user>
              <res-password></res-password>
           </default-mapping>
           <principal-mapping-entry>
              <initiating-user>servletuser</initiating-user>
              <res-user>jmsuser</res-user>
              <res-password>->jmsuser</res-password>
           </principal-mapping-entry>
        </principal-mapping-entries>
     </security-config>
     <connectionfactory-interface>
        javax.jms.XAQueueConnectionFactory
     </connectionfactory-interface>
  </connector-factory>
  <connector-factory location="OracleASjms/MyCF1" connector-name="OracleASjms">
     <config-property name="jndiLocation" value="jms/ConnectionFactory1"/>
     <connection-pooling use="shared">
        <use-connection-pool>commonPool</use-connection-pool>
     </connection-pooling>
     <connectionfactory-interface>
        javax.jms.ConnectionFactory
     </connectionfactory-interface>
  </connector-factory>
  <connector-factory location="OracleASjms/MyCF2" connector-name="OracleASjms">
     <config-property name="jndiLocation" value="jms/ConnectionFactory2"/>
     <connection-pooling use="shared">
        <use-connection-pool>commonPool</use-connection-pool>
     </connection-pooling>
     <connectionfactory-interface>
        javax.jms.ConnectionFactory
     </connectionfactory-interface>
  </connector-factory>
  <connection-pool name="commonPool">
     <property name="minConnections" value="0" />
     <property name="maxConnections" value="10" />
     <property name="waitTimeout" value="300" />
     <property name="scheme" value="fixed_wait" />
  </connection-pool>
</oc4j-connector-factories>
```

Hierarchy of oc4j-connectors.xml

Here is an overview of the `oc4j-connectors.xml` element hierarchy.

```
<oc4j-connectors>
   <connector>
      <config-property>
      <security-permission>
         <security-permission-spec>
      <adminobject-config>
         <adminobject-class>
         <config-property>
      <oc4j-connectors>
      <start-order>
```

Elements and Attributes of oc4j-connectors.xml

This section is an alphabetical dictionary of elements of the `oc4j-connectors.xml` file. See the preceding section, "Hierarchy of oc4j-connectors.xml", if you are interested in the hierarchy.

<adminobject-class>

Parent element: <adminobject-config>

Child elements: None

Required? Required; one only

The value of this element indicates the fully qualified name of the JavaBean class for the administered object:

```
<adminobject-config>
   <adminobject-class>classname</adminobject-class>
   ...
</adminobject-config>
```

The `<adminobject-class>` element has no attributes.

<adminobject-config>

Parent element: <connector>

Child elements: <adminobject-class>, <config-property>

Required? Optional; zero or more

Each `<adminobject-config>` element is for an administered object for the resource adapter, specifying through its attribute and subelements the JNDI location, administered object class, and any configuration property settings.

See Chapter 7, "Managing Administered Objects" for general information about administered objects and additional information about configuring them.

Table A–11 <adminobject-config> Attributes

Name	Description
location	Values: String Default: n/a (required) The JNDI location to which the administered object instance will be bound.

<config-property>

Parent element: <adminobject-config>

Child elements: None

Required? Optional; zero or more

Each `<config-property>` subelement of an `<adminobject-config>` element specifies the name and desired value of a property of the administered object JavaBean.

Table A–12 <config-property> Attributes

Name	Description
name	Values: String Default: n/a (required) The name of a property of the administered object JavaBean.
value	Values: String Default: n/a (required) The desired value of the administered object property.

<config-property>

Parent element: <connector>

Child elements: None

Required? Optional; zero or more

This element, when a subelement of a `<connector>` element, indicates the name and deployed value of a property of the resource adapter JavaBean. Each `<config-property>` subelement of a `<connector>` element corresponds to a `<config-property>` subelement of a `<resourceadapter>` element in the `ra.xml` file, where the `ra.xml` file specifies the name and assembled value of the property.

See "Setting Properties of the Resource Adapter JavaBean" on page 2-7 for related information.

Table A–13 <config-property> Attributes

Name	Description
name	Values: String Default: n/a (required) The name of a resource adapter property (from `ra.xml`)

Table A–13 (Cont.) <config-property> Attributes

Name	Description
value	Values: String Default: n/a (required) The desired deployed value of the property (can override the assembled value from `ra.xml`).

<connector>

Parent element: <oc4j-connectors>

Child elements: <config-property>, <security-permission>, <adminobject-config>, <native-library>, <start-order>

Required? Optional; zero or more

There is a `<connector>` element for each resource adapter deployed within the EAR file (for resource adapters deployed with an application), or for each standalone resource adapter (in the case of the `oc4j-connectors.xml` file associated with the OC4J default application). This element indicates the name, path, and JNDI location of the corresponding resource adapter, along with additional configuration through its subelements.

Table A–14 <connector> Attributes

Name	Description
name	Values: String Default: n/a (required) The name of the applicable resource adapter, as determined during deployment for a standalone resource adapter (such as through the Application Server Control Console deployment page), or according to the RAR file name, without the `.rar` extension, for a resource adapter deployed within an EAR file. This corresponds to the `connector-name` attribute of applicable `<connector-factory>` elements in the `oc4j-ra.xml` file.
path	Values: String Default: n/a (required) The directory path from which the RAR file was deployed.
location	Values: String Default: Empty string A JNDI location to which the resource adapter instance will be bound. This is relevant only if the resource adapter supports message inflow, and is generally for internal use only.

<native-library>

Parent element: <connector>

Child elements: None

Required? Optional; zero or one

This element points to the location of a native library for use by the resource adapter (and provided with the resource adapter). Also see "Configuring the Use of Resource Adapter Native Libraries" on page 2-8.

Table A–15 <native-library> Attributes

Name	Description
path	Values: String Default: No default The relative path for a native library, according to its relative location in the RAR file during deployment.

Note: You can use either the path attribute or the element value to specify the path. Either of the following examples is valid:

```
<native-library path="lib/foo.so" />

<native-library>lib/foo.so</native-library>
```

<oc4j-connectors>

Parent element: n/a (root)

Child elements: <connector>

Required? Required; one only

This is the top-level element of the `oc4j-connectors.xml` file, encompassing the enumeration of all resource adapters deployed within a single EAR file, or all standalone resource adapters, as applicable.

Table A–16 <oc4j-connectors> Attributes

Name	Description
schema-major-version	Values: String Default: No default The major version number of the `oc4j-connectors.xml` XSD. If you create `oc4j-connectors.xml` manually, set this attribute to 10 for use with the OC4J 10.1.3 implementation.
schema-minor-version	Values: String Default: No default The minor version number of the `oc4j-connectors.xml` XSD. If you create `oc4j-connectors.xml` manually, set this attribute to 0 for use with the OC4J 10.1.3 implementation.

Note: These attributes do not appear directly in the XSD for `ocj4-connectors.xml`. They are according to the `attributeGroup` specification in the top-level OC4J XSD.

<security-permission>

Parent element: <connector>

Child elements: <security-permission-spec>

Required? Optional; zero or more

Each <security-permission> element corresponds to a <security-permission> element in the ra.xml file, and gives the deployer the ability to grant or deny the list of permissions requested by the resource adapter. (See "Security Permissions" on page 1-7 for related information.)

Table A–17 <security-permission> Attributes

Name	Description
enabled	Values: Boolean Default: false Set this to "true" to enable the security specification indicated in the <security-permission-spec> subelement.

<security-permission-spec>

Parent element: <security-permission>

Child elements: None

Required? Required within <security-permission>; one only

The value of this element (which may contain multiple lines of instructions) specifies a security permission based on security policy file syntax. Refer to the following URL for the Sun Microsystems implementation of the security permission specification:

http://java.sun.com/j2se/1.4.2/docs/guide/security/PolicyFiles.html#FileSyntax

See version 1.5 of the *J2EE Connector Architecture Specification* for additional information and examples.

The <security-permission-spec> element has no attributes.

<start-order>

Parent element: <connector>

Child elements: None

Required? Optional; zero or one

The values of any <start-order> elements in an oc4j-connectors.xml file determine the order in which resource adapters are loaded and started by OC4J within the application. In the case of the oc4j-connectors.xml file for the OC4J default application, this is the order in which standalone resource adapters are loaded and started.

<start-order>*value*</start-order>

There can be one (optional) element, under the associated <connector> element, for each resource adapter. The value must be a positive integer. Resource adapters with lower values are loaded and started first. If no value is specified for a resource adapter, or if the value specified is not a positive integer, OC4J is free to load the resource adapter according to any sequence.

See "Understanding the Resource Adapter Startup Process" on page 2-12 for related information.

The <start-order> element has no attributes.

Sample oc4j-connectors.xml

This section shows a sample oc4j-connectors.xml file that lists a data source resource adapter and a JMS resource adapter, including configuration of administered objects for topics and queues for the JMS adapter.

```
<?xml version="1.0" standalone='yes'?>

<oc4j-connectors xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation=
 "http://xmlns.oracle.com/oracleas/schema/oc4j-connectors-10_0.xsd"
 schema-major-version="10" schema-minor-version="0"  >
   <connector name="datasources" path="datasources.rar" location="datasources" >
   </connector>
   <connector name="OracleASjms" path="OracleASjms.rar" location="OracleASjms" >
      <config-property name="lookupMethod" value="resourceProvider"/>
      <config-property name="resourceProviderName" value="oc4jjms"/>
      <adminobject-config location="OracleASjms/MyQueue1">
         <adminobject-class>
            oracle.j2ee.ra.jms.generic.AdminObjectQueueImpl
         </adminobject-class>
         <config-property name="jndiName" value="jms/demoQueue"/>
         <config-property name="resourceProviderName" value="oc4jjms"/>
      </adminobject-config>
      <adminobject-config location="OracleASjms/MyTopic1">
         <adminobject-class>
            oracle.j2ee.ra.jms.generic.AdminObjectTopicImpl
         </adminobject-class>
         <config-property name="jndiName" value="jms/demoTopic"/>
         <config-property name="resourceProviderName" value="oc4jjms"/>
      </adminobject-config>
   </connector>
</oc4j-connectors>
```

B

Third Party Licenses

This appendix includes the Third Party License for all the third party products included with Oracle Application Server.

ANTLR

This program contains third-party code from ANTLR. Under the terms of the Apache license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the ANTLR software, and the terms contained in the following notices do not change those rights.

The ANTLR License

```
Software License

We reserve no legal rights to the ANTLR--it is fully in the public domain.
An individual or company may do whatever they wish with source code distributed
with ANTLR or the code generated by ANTLR, including the incorporation of
ANTLR, or its output, into commerical software.

We encourage users to develop software with ANTLR. However, we do ask that
credit is given to us for developing ANTLR. By "credit", we mean that if you
use ANTLR or incorporate any source code into one of your programs (commercial
product, research project, or otherwise) that you acknowledge this fact
somewhere in the documentation, research report, etc... If you like ANTLR and
have developed a nice tool with the output, please mention that you developed
it using ANTLR. In addition, we ask that the headers remain intact in our
source code. As long as these guidelines are kept, we expect to continue
enhancing this system and expect to make other tools available as they are
completed.
```

Apache

This program contains third-party code from the Apache Software Foundation ("Apache"). Under the terms of the Apache license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights.

The Apache license agreements apply to the following included Apache components:

- Apache HTTP Server
- Apache JServ

- mod_jserv
- Regular Expression package version 1.3
- Apache Expression Language packaged in commons-el.jar
- mod_mm 1.1.3
- Apache XML Signature and Apache XML Encryption v. 1.4 for Java and 1.0 for C++
- log4j 1.1.1
- BCEL v. 5
- XML-RPC v. 1.1
- Batik v. 1.5.1
- ANT 1.6.2 and 1.6.5
- Crimson v. 1.1.3
- ant.jar
- bcel.jar
- soap.jar
- Jakarta CLI 1.0
- jakarta-regexp-1.3.jar
- JSP Standard Tag Library 1.0.6 and 1.1
- Struts 1.1
- Velocity 1.3
- svnClientAdapter
- commons-logging.jar
- wsif.jar
- commons-el.jar
- standard.jar
- jstl.jar

The Apache Software License

License for Apache Web Server 1.3.29

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000-2002 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
```

```
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing
Applications,
 * University of Illinois, Urbana-Champaign.
```

License for Apache Web Server 2.0

```
Copyright (c) 1999-2004, The Apache Software Foundation
Licensed under the Apache License, Version 2.0 (the "License"); you may not use
this file except in compliance with the License.  You may obtain a copy of the
License at   ://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed
under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
 CONDITIONS OF ANY KIND, either express or implied.  See the License for the
 specific language governing permissions and limitations under the License.
Copyright (c) 1999-2004, The Apache Software Foundation
                                 Apache License
                            Version 2.0, January 2004
                         http://www.apache.org/licenses/
```

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and

may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

Apache SOAP

This program contains third-party code from the Apache Software Foundation ("Apache"). Under the terms of the Apache license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights.

Notwithstanding anything to the contrary in the Oracle program license, the Apache software is provided by Oracle "AS IS" and without warranty or support of any kind from Oracle or Apache.

Apache SOAP License

Apache SOAP license 2.3.1

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems,

and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and

do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Index

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

W

X