BEA WebLogic Server™

Programming the WebLogic J2EE Connector Architecture

BEA WebLogic Server 6.1
Document Date: June 24, 2002

Programming the WebLogic J2EE Connector Architecture

Part Number	Document Date	Software Version
N/A	June 24, 2002	BEA WebLogic Server Version 6.1

Contents

About This Document

This document introduces the WebLogic J2EE Connector Architecture and describes how to configure and deploy resource adapters to WebLogic Server. The document is organized as follows:

- Chapter 1, "Overview of the WebLogic J2EE Connector Architecture," provides an overview of the WebLogic J2EE Connector Architecture.
- Chapter 2, "Security," discusses WebLogic J2EE Connector Architecture security considerations.
- Chapter 3, "Transaction Management," introduces the various types of transaction levels supported by the WebLogic J2EE Connector Architecture and explains how to specify the transaction levels in the resource adapter `.rar` archive.
- Chapter 4, "Connection Management," introduces you to various connection management tasks.
- Chapter 5, "Configuration," outlines the configuration tasks that you perform to deploy resource adapters to WebLogic Server.
- Chapter 6, "Writing J2EE Connector Architecture- Compliant Resource Adapters," provides requirements for writing a resource adapter (`.rar`).
- Chapter 7, "Resource Adapter Deployment," provides an overview of resource adapters and explains how to configure and deploy them to WebLogic Server.
- Chapter 8, "Client Considerations," discusses WebLogic J2EE Connector Architecture client considerations.
- Appendix A, "weblogic-ra.xml Deployment Descriptor Elements," provides the `weblogic-ra.xml` DTD and deployment descriptor elements.

- Appendix B, "Workarounds for Common BEA J2EE Connector Architecture Exceptions," provides troubleshooting information for two common connector exceptions.

Audience

This document is written for application developers who want to build e-commerce applications using the Java 2 Platform, Enterprise Edition (J2EE) from Sun Microsystems. It is assumed that readers know Web technologies, object-oriented programming techniques, and the Java programming language.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. In particular, refer to the following:

- Javadoc for the BEA WebLogic J2EE Connector Architecture (See the product distribution CD.)
- Weblogic-specific Resource Adapter Document Type Definition (See Appendix A, "weblogic-ra.xml Deployment Descriptor Elements.")
- BEA WebLogic Application Integration (See http://edocs.bea.com/wlintegration/v2_0/applicationintegration/devel/index.htm.) This document describes how to build a resource adapter.

Also refer to the following documentation from Sun Microsystems:

- J2EE Connector Architecture—http://java.sun.com/j2ee/connector/index.html
- J2EE Connector Specification, Version 1.0, Proposed Final Draft 2—http://java.sun.com/j2ee/download.html#connectorspec
- J2EE Platform Specification, Version 1.3, Proposed Final Draft 3—http://java.sun.com/j2ee

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA

WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
italics	Emphasis and book titles.

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

Convention	Usage
`monospace italic text`	Variables in code. *Example*: `String CustomerName;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.

`[ ]`	Optional items in a syntax line. *Example*:
	`java utils.MulticastTest -n name -a address`
	`     [-p portnumber] [-t timeout] [-s send]`

`|`	Separates mutually exclusive choices in a syntax line. *Example*:
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`     password {application} {source}`

`...`	Indicates one of the following in a command line: ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information
. . .	Indicates the omission of items from a code example or from a syntax line.

CHAPTER

1 Overview of the WebLogic J2EE Connector Architecture

The following sections provide an overview of the BEA WebLogic J2EE Connector Architecture:

- WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality
- Overview of the BEA WebLogic J2EE Connector Architecture Implementation
- J2EE Connector Architecture Components
- Black Box Example

WebLogic Server 6.1 with J2EE 1.2 and J2EE 1.3 Functionality

This document illustrates functionality defined in a non-final specification, J2EE 1.3. It requires you to run WebLogic Server with J2EE 1.3 features enabled. If you attempt to deploy the WebLogic Server Connector Architecture with the WebLogic Server 6.1 distribution that has J2EE 1.2 features only, the deployment will fail.

Both the J2EE 1.2-certified distribution of WebLogic Server 6.1 with J2EE 1.2 features only and the WebLogic Server 6.1 distribution with J2EE 1.3 features enabled are available at http://commerce.bea.com/downloads/products.jsp. They are also available on the Product CD.

J2EE Connector Architecture Terminology

Key terms and concepts that you will encounter throughout the WebLogic J2EE Connector Architecture documentation include the following:

- Common client interface (CCI)—defines a standard client API for application components and enables application components and Enterprise Application Integration (EAI) frameworks to drive interactions across heterogeneous EISes using a common client API. The J2EE Connector Architecture defines a CCI for EIS access.
- Container—part of an application server—such as WebLogic Server—that provides deployment and run-time support for application components. A container allows you to monitor and manage supported components as well as the service(s) that monitor and manage the components. Containers can be one of the following:
  - Connector containers that host resource adapters
  - Web containers that host JSP, servlets, and static HTML pages
  - EJB containers that host EJB components

- Application client containers that host standalone application clients

  For more details on different types of standard containers, refer to Enterprise JavaBeans (EJBs), Java Server Pages (JSPs), and Servlets specifications.

- Enterprise Information System (EIS) resource—provides EIS-specific functionality to its clients. Examples are:
  - Record or set of records in a database system
  - Business object in an Enterprise Resource Planning (ERP) System
  - Transaction program in a transaction processing system
- Enterprise Information System (EIS)—provides the information infrastructure for an enterprise. An EIS offers a set of services to its clients. These services are exposed to clients as local and/or remote interfaces. Examples of an EIS include:
  - ERP system
  - Mainframe transaction processing system
  - Legacy database system
- J2EE Connector—See Resource Adapter.
- J2EE Connector Architecture—an architecture for integration of J2EE-compliant application servers with enterprise information systems (EISes). There are two parts to this architecture: an EIS vendor-provided resource adapter and an application server—such as WebLogic Server— to which the resource adapter plugs in. This architecture defines a set of contracts—such as transactions, security, and connection management—that a resource adapter has to support to plug in to an application server. The J2EE Connector Architecture also defines a Common Client Interface (CCI) for EIS access. The CCI defines a client API for interacting with heterogeneous EISes.
- Managed environment—defines an operational environment for a J2EE-based, multi-tier, Web-enabled application that accesses EISes. The application consists of one or more application components—EJBs, JSPs, servlets—which are deployed on containers. These containers can be one of the following:
  - Web containers that host JSP, servlets, and static HTML pages
  - EJB containers that host EJB components
  - Application client containers that host standalone application clients

- Non-managed environment—defines an operational environment for a two-tier application. An application client directly uses a resource adapter to access the EIS; the EIS defines the second tier for a two-tier application.
- `.rar` file—resource adapter archive. A compressed (zip) file used to load classes and other files required to run a resource adapter.
- `ra.xml` file—describes the resource adapter-related attributes type and its deployment properties using a standard DTD from Sun Microsystems.
- Resource Adapter—a system-level software driver used by an application server such as WebLogic Server to connect to an EIS. A resource adapter serves as the "J2EE connector." The WebLogic J2EE Connector Architecture supports resource adapters developed by Enterprise Information Systems (EISes) vendors and third-party application developers that can be deployed in any application server supporting the Sun Microsystems J2EE Platform Specification, Version 1.3. Resource adapters contain the Java, and if necessary, the native components required to interact with the EIS.
- Resource manager—part of an EIS that manages a set of shared EIS resources. Examples of resource managers are a database system, a mainframe TP system, and an ERP system. A client requests access to a resource manager to use its managed resources. A transactional resource manager can participate in transactions that are externally controlled and coordinated by a transaction manager. In the context of the J2EE Connector Architecture, clients of a resource manager can include middle-tier application servers and client-tier applications. A resource manager is typically a different address space or on a different machine from the client that accesses it.
- Service provider interface (SPI)—contains the objects that provide and manage connectivity to the EIS, establish transaction demarcation, and provide a framework for event listening and request transmission. All J2EE Connector Architecture-compliant resource adapters must provide an implementation for these interfaces in the `javax.resource.spi` package.
- System contract—a mechanism by which connection requests are passed between entities. To achieve a standard system-level pluggability between application servers such as WebLogic Server and EISes, the Connector Architecture defines a standard set of system-level contracts between an application server and an EIS. The EIS side of these system-level contracts is implemented in a resource adapter.

- `weblogic-ra.xml` file—adds additional WebLogic Server-specific deployment information to the `ra.xml` file.

Overview of the BEA WebLogic J2EE Connector Architecture Implementation

BEA WebLogic Server continues to build upon the implementation of the Sun Microsystems J2EE Platform Specification, Version 1.3. The J2EE Connector Architecture adds simplified Enterprise Information System (EIS) integration to the J2EE platform. The goal is to leverage the strengths of the J2EE platform—including component models, transaction and security infrastructures—to address the challenges of EIS integration.

The J2EE Connector Architecture provides a Java solution to the problem of connectivity between the multitude of application servers and EISes. By using the Connector Architecture, it is no longer necessary for EIS vendors to customize their product for each application server. By conforming to the J2EE Connector Architecture, BEA WebLogic Server does not require added custom code in order to extend its support connectivity to a new EIS.

The Connector Architecture enables an EIS vendor to provide a standard resource adapter for its EIS. This resource adapter plugs into WebLogic Server and provides the underlying infrastructure for the integration between an EIS and WebLogic Server.

By supporting the Connector Architecture, BEA WebLogic Server is assured of connectivity to multiple EISes. In turn, EIS vendors must provide only one standard Connector Architecture-compliant resource adapter that has the capability to plug into BEA WebLogic Server.

J2EE Connector Architecture Components

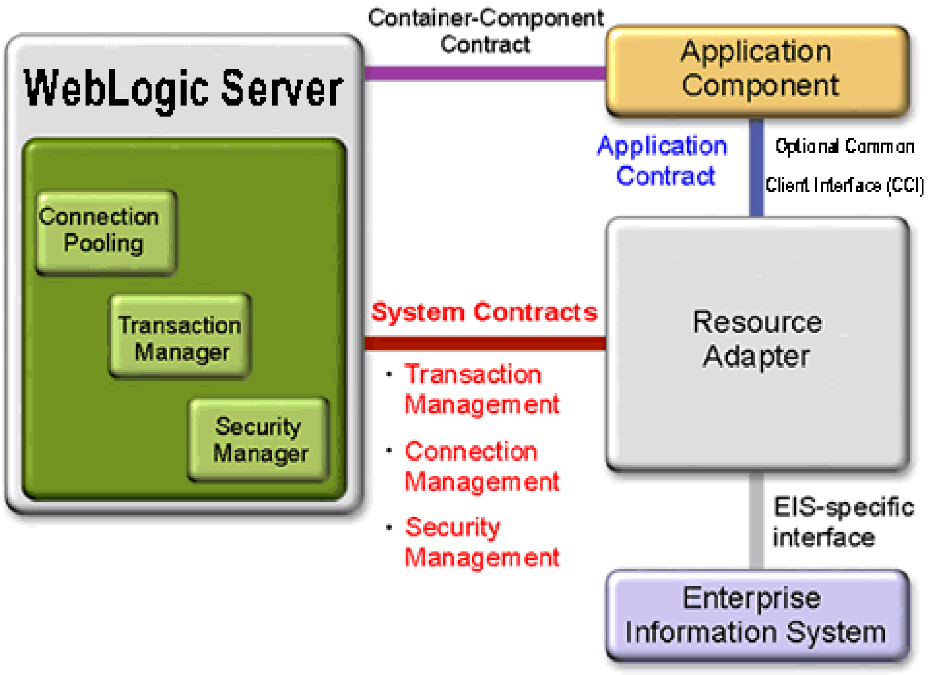
The J2EE Connector Architecture is implemented in an application server such as WebLogic Server and an EIS-specific resource adapter. A resource adapter is a system library specific to an EIS and provides connectivity to the EIS. A resource adapter is analogous to a JDBC driver. The interface between a resource adapter and the EIS is specific to the underlying EIS; it can be a native interface.

The J2EE Connector Architecture has three main components:

- System-level Contracts—between the resource adapter and the application server (WebLogic Server)
- Common Client Interface (CCI)—provides a client API for Java applications and development tools to access the resource adapter
- Packaging and Deployment Interfaces—provides ability for various resource adapters to plug into J2EE applications in a modular manner

The following diagram illustrates the J2EE Connector Architecture:

Figure 1-1 J2EE Connector Architecture



A resource adapter serves as the "J2EE connector." The WebLogic J2EE Connector Architecture supports resource adapters developed by Enterprise Information Systems (EISes) vendors and third-party application developers that can be deployed in any application server supporting the Sun Microsystems J2EE Platform Specification, Version 1.3. Resource adapters contain the Java, and if necessary, the native components required to interact with the EIS.

System-level Contracts

The J2EE Connector Architecture specification defines a set of system-level contracts between the J2EE-compliant application server (WebLogic Server) and an EIS-specific resource adapter. WebLogic Server, in compliance with this specification, has implemented a set of defined standard contracts for:

- Connection management—a contract that gives an application server pool connections to underlying EISes. It also allows application components to connect to an EIS. This results in a scalable application environment that supports a large number of clients requiring access to EISes.

Note: For more information on connection management, refer to

- Transaction management—a contract between the transaction manager and an EIS supporting transaction access to EIS resource managers. This contract allows an application server to use a transaction manager to manage transactions across multiple resource managers.

Note: For more information on transaction management, refer to Chapter 3, “Transaction Management.”

- Security management—a contract that provides secure access to an EIS and provides support for a secure application environment. This reduces threats to the EIS and protects information resources that the EIS manages.

Note: For more information on security management, refer to Chapter 2, “Security.”

Common Client Interface (CCI)

The Common Client Interface (CCI) defines a standard client API for application components. The CCI enables application components and Enterprise Application Integration (EAI) frameworks to drive interactions across heterogeneous EISes using a common client API.

The target users of the CCI are enterprise tool vendors and EAI vendors. Application components themselves may also write to the API, but the CCI is a low-level API. The specification recommends that the CCI be the basis for richer functionality provided by the tool vendors, rather than being an application-level programming interface used by most application developers.

Further, the CCI defines a remote function-call interface that focuses on executing functions on an EIS and retrieving the results. The CCI is independent of a specific EIS; for example: data types specific to an EIS. However, the CCI is capable of being driven by EIS-specific metadata from a repository.

The CCI enables WebLogic Server applications to create and manage connections to an EIS, execute an interaction, and manage data records as input, output or return values. The CCI is designed to leverage the JavaBeans architecture and Java Collection framework.

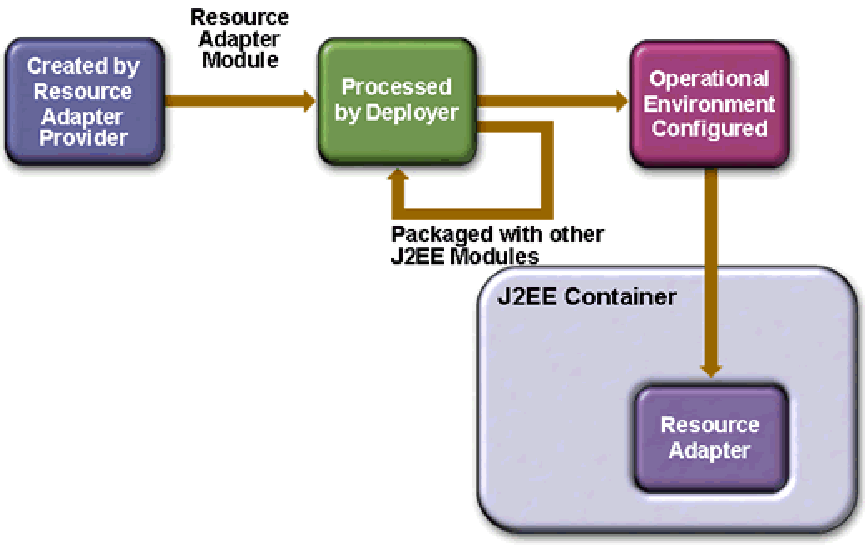
The 1.0 version of the J2EE Connector Architecture recommends that a resource adapter support CCI as its client API, while it requires that the resource adapter implement the system contracts. A resource adapter may choose to have a client API different from CCI, such as the client API based on the Java Database Connectivity (JDBC) API.

Note: For more information relating to the Common Client Interface, refer to Chapter 8, "Client Considerations."

Packaging and Deployment

The J2EE Connector Architecture provides packaging and deployment interfaces, so that various resources adapters can easily plug into compliant J2EE application servers such as WebLogic Server in a modular manner.

Figure 1-2 Packaging and Deployment


Created by Resource Adapter Provider
Resource Adapter Module
Processed by Deployer
Operational Environment Configured
Packaged with other J2EE Modules
J2EE Container
Resource Adapter

A resource adapter provider develops a set of Java interfaces and classes as part of its implementation of a resource adapter. These Java classes implement J2EE Connector Architecture-specified contracts and EIS-specific functionality provided by the resource adapter. The development of a resource adapter can also require use of native libraries specific to the underlying EIS.

The Java interfaces and classes are packaged together (with required native libraries, help files, documentation, and other resources) with a deployment descriptor to create a Resource Adapter Module. A deployment descriptor defines the contract between a resource adapter provider and a deployer for the deployment of a resource adapter.

You can deploy resource adapter module as a shared, stand-alone module or packaged as part of an application. During deployment, you install a resource adapter module on an application server such as WebLogic Server and then configure it into the target operational environment. The configuration of a resource adapter is based on the properties defined in the deployment descriptor as part of the resource adapter module.

Note: For more information on packaging and deployment, refer to Chapter 6, “Writing J2EE Connector Architecture- Compliant Resource Adapters,” Chapter 5, “Configuration,” and Chapter 7, “Resource Adapter Deployment.”

Black Box Example

A simple code example for a resource adapter is provided with this release. This code example uses a Black Box resource adapter that mimics JDBC calls. An EJB is used to model the data in the Black Box, and a Java client is used to query the Black Box resource adapter and display the output. The example uses the all-Java Cloudscape DBMS, which is provided in an evaluation version with WebLogic Server. For more information, refer to the WebLogic J2EE Connector Architecture Example Javadoc provided with the product download.

CHAPTER

2 Security

The following sections discuss WebLogic J2EE Connector Architecture security:

- Container-Managed and Application-Managed Sign-on
- Security Principal Map
- Password Converter Tool

Container-Managed and Application-Managed Sign-on

As specified in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2, the WebLogic J2EE Connector Architecture implementation supports both container-managed and application-managed sign-on.

At runtime, the Weblogic J2EE Connector Architecture implementation determines—based upon the specified information in the invoking client component's deployment descriptor—the chosen sign-on mechanism. If the Weblogic Server J2EE Connector Architecture implementation is unable to determine what sign-on mechanism is being requested by the client component—typically due to an improper JNDI lookup of the resource adapter Connection Factory—the Connector Architecture attempts container-managed sign-on.

Note: Note that even in this case, if the client component has specified explicit security information, this information is also presented on the call to obtain the connection.

For related information, see "Obtaining the ConnectionFactory (Client-JNDI Interaction)" in Chapter 8, "Client Considerations."

Application-Managed Sign-on

With application-managed sign-on, the client component provides the necessary security information (typically a username and password) when making the call to obtain a connection to an Enterprise Information System (EIS). In this scenario, the application server provides no additional security processing other than to pass this information along on the request for the connection. The provided resource adapter uses the client component provided security information to perform the EIS sign-on in a resource adapter implementation specific manner.

Container-Managed Sign-on

With container-managed sign-on, the client component does not present any security information, and the container must determine the necessary sign-on information and provide this information to the resource adapter when making a call to request a connection. In all container-managed sign-on scenarios, the container must determine an appropriate Resource Principal and provide this Resource Principal information to the resource adapter in the form of a Java Authentication and Authorization Service (JAAS) Subject.

Security Principal Map

The "EIS Sign-on" section of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec) identifies a number of possible options for defining a Resource Principal on whose behalf the sign-on is being performed. The Weblogic Server implementation implements the Security Principal Map option identified in the specification.

Under this option, a resource principal is determined by mapping from the identity of the initiating/caller principal for the invoking component. The resultant resource principal does not inherit the identity or security attributes of the principal that it is mapped from, but instead gets its identity and security attributes (password) based upon the defined mapping.

Therefore, in order to enable and use container-managed sign-on, Weblogic Server must provide a mechanism to specify the `initiating-principal` to `resource-principal` association. WebLogic Server does this through a Security Principal Map that can be defined for each deployed resource adapter.

If container-managed sign-on is requested by the client component and no Security Principal Map is configured for the deployed resource adapter, an attempt is made to obtain the connection, but the provided JAAS Subject will be `NULL`. Support for this scenario will be based upon the resource adapter implementation.

A scenario in which omitting configuration of a Security Principal Map might be considered valid is the case in which a resource adapter internally obtains all of its EIS connections with a hard-coded and pre-configured set of security information, and therefore does not depend on the security information passed to it on requests for new connections. (In a sense, this is a third scenario, outside of application-managed sign-on and container-managed sign-on.)

While the defined connection management system contracts define how security information is exchanged between WebLogic Server and the provided resource adapter, the determination of whether to use container-managed sign-on or application-managed sign-on is based on deployment information defined for the client application that is requesting a connection. For more information on how a connection management system contract is specified, see Chapter 8, “Client Considerations.”

For more information on how client components specify the sign-on mechanism, see the “Application Programming Model” section of the “Connection Management” chapter in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec).

For more information on the J2EE Connector Architecture application security model, see the “Application Security Model” of the same document.

Using Container-Managed Sign-On

To use container-managed sign-on, WebLogic Server must identify a resource principal and then request the connection on behalf of the resource principal. In order to make this identification, WebLogic Server looks for a Security Principal Mapping specified with the `security-principal-map` element in the `weblogic-ra.xml` deployment descriptor file.

A `security-principal-map` element defines the relationship of `initiating-principal` to a `resource-principal`.

Each `security-principal-map` element provides a mechanism to define appropriate resource principal values for resource adapter and EIS sign-on processing. The `security-principal-map` elements allow you to specify a defined set of initiating principals and the corresponding resource principal's username and password to be used when allocating managed connections and connection handles.

Default Resource Principal

A default resource principal can be defined for the connection factory in the `security-principal-map` element. If you specify an `initiating-principal` value of `'*'` and a corresponding `resource-principal` value, the defined `resource-principal` is utilized whenever the current identity is *not* matched elsewhere in the map.

This is an optional element, however. You must specify it in some form if container-managed sign-on is supported by the resource adapter and used by *any* client.

In addition, the deployment-time population of the Connection Pool with Managed Connections is attempted using the defined 'default' resource principal if one is specified.

For instructions on configuring the J2EE Connector Architecture `security-principal-map` and associating it with the deployed `.rar` (resource adapter), refer to “Configuring the Security Principal Map” in Chapter 5, “Configuration.”

Password Converter Tool

Because BEA understands the importance of protecting security passwords, it provides a converter tool that can encrypt all passwords present in the `weblogic-ra.xml` file.

For more information, refer to “Configuring the Security Principal Map,” in Chapter 5, “Configuration.”

CHAPTER

3 Transaction Management

The following sections describe the various types of transaction levels supported by the WebLogic J2EE Connector Architecture and explain how to specify the transaction levels in the resource adapter `.rar` archive.

- Supported Transaction Levels
- Specifying the Transaction Levels in the .rar Configuration
- Transaction Management Contract

Supported Transaction Levels

Transactional access to EISes is an important requirement for business applications. The J2EE Connector Architecture supports the concept of transactions—a number of operations that must be committed together or not at all for the data to remain consistent and to maintain data integrity.

The BEA WebLogic Server J2EE Connector Architecture implementation utilizes WebLogic Server's robust Transaction Manager implementation and supports resource adapters having the following transaction support levels (as described in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2):

- XA Transaction support—allows a transaction to be managed by a transaction manager external to a resource adapter (and therefore external to an EIS). A resource adapter defines the type of transaction support by specifying the transaction-support element in the `ra.xml` file; a resource adapter can only support one type. When an application component demarcates an EIS connection request as part of a transaction, the application server is responsible for enlisting the XA resource with the transaction manager. When the application component closes that connection, the application server de-lists the XA resource from the transaction manager and cleans up the EIS connection once the transaction has completed.
- Local Transaction support—allows an application server to manage resources, which are local to the resource adapter. Unlike XA transaction, it cannot participate in a two-phase commit protocol (2PC). A resource adapter defines the type of transaction support by specifying the transaction-support element in the resource adapter `ra.xml` file; a resource adapter can only support one type. When an application component requests for an EIS connection, the application server starts a local transaction based on the current transaction context. When the application component closes that connection, the application server does a commit on the local transaction and also cleans up the EIS connection once the transaction has completed.

  Note: Refer to the following Sun Microsystems documentation for information on the `ra.xml` document type definition: http://java.sun.com/dtd/connector_1_0.dtd

- No Transaction support—in general, if a resource adapter does not support XA or Local Transaction support (and therefore "supports" No Transaction), it means that if an application component needs to use that resource adapter, the application component must not involve any connections to the EIS, represented by that resource adapter, in a transaction. However, if an application component needs to involve EIS connections in a transaction, the application component *must* interact with a resource adapter that supports XA or Local Transactions.

For more information on supported transaction levels, see the "Transaction Management" chapter in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec).

Specifying the Transaction Levels in the .rar Configuration

The resource adapter specifies which kind of transaction it supports in the `ra.xml` deployment descriptor file provided by Sun Microsystems. For instructions on specifying the transaction level type in the `.rar`, refer to "Configuring the Transaction Level Type" in Chapter 5, "Configuration."

Note: Refer to the following Sun Microsystems documentation for information on the `ra.xml` document type definition: http://java.sun.com/dtd/connector_1_0.dtd

Transaction Management Contract

In many cases, a transaction (termed local transaction) is limited in scope to a single EIS system, and the EIS resource manager itself manages such a transaction. While an XA transaction (or global transaction) can span multiple resource managers. This form of transaction requires transaction coordination by an external transaction manager, typically bundled with an application server. A transaction manager uses a two-phase commit protocol (2PC) to manage a transaction that spans multiple resource managers (EISes). It uses one-phase commit optimization if only one resource manager is participating in an XA transaction.

The J2EE Connector Architecture defines a transaction management contract between an application server and a resource adapter (and its underlying resource manager). The transaction management contract extends the connection management contract and provides support for management of both local and XA transactions. The transaction management contract has two parts, depending on the type of transaction.

- JTA XAResource based contract between a transaction manager and an EIS resource manager
- Local transaction management contract

These contracts enable an application server such as WebLogic Server to provide the infrastructure and runtime environment for transaction management. Application components rely on this transaction infrastructure to support the component-level transaction model.

Because EIS implementations are so varied, the transactional support must be very flexible. The J2EE Connector Architecture imposes no requirements on the EIS for transaction management. Depending on the implementation of transactions within the EIS, a resource adapter may provide:

- No transaction support at all—this is typical of legacy applications and many back-end systems.
- Support for only local transactions
- Support for both local and XA transactions

WebLogic Server supports all three levels of transactions, ensuring its support of EISes at different transaction levels.

CHAPTER

4 Connection Management

The following sections introduce you to the various connection management tasks relating to the BEA WebLogic J2EE Connection Management Architecture.

- Error Logging and Tracing Facility
- Configuring Connection Properties
- BEA WebLogic Server Extended Connection Management Features
- Monitoring Connection Pools Using the Console

Error Logging and Tracing Facility

As stated in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2, one of the requirements for application servers is use of `ManagedConnectionFactory.set/getLogWriter` to provide an error logging and tracing facility for the resource adapter.

Two elements are provided in the `weblogic-ra.xml` descriptor file to configure this feature in BEA WebLogic Server:

- The `logging-enabled` element indicates whether logging is turned on or off. The default value for this element is `false`.
- The `log-filename` element specifies the filename in which to write the logging information.

For more information, see Appendix A, "weblogic-ra.xml Deployment Descriptor Elements."

Configuring Connection Properties

The `ra.xml` deployment descriptor file contains a `config-property` element to declare a single configuration setting for a `ManagedConnectionFactory` instance. The resource adapter provider typically sets these configuration properties. However, if a configuration property is not set, the person deploying the resource adapter is responsible for providing a value for the property.

WebLogic Server allows you to set configuration properties through the use of the `map-config-property` element in the `weblogic-ra.xml` deployment descriptor file. To configure a set of configuration properties for a resource adapter, you specify a `map-config-property-name` and `map-config-property-value` pair for each configuration property to declare.

You can also use the `map-config-property` element to override the values specified in the `ra.xml` deployment descriptor file. At startup, WebLogic Server compares the values of `map-config-property` against the values of `config-property` in the `ra.xml` file. If the configuration property names match, WebLogic Server uses the `map-config-property-value` for the corresponding configuration property name.

BEA WebLogic Server Extended Connection Management Features

In addition to the connection management requirements stated in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2, BEA WebLogic Server provides optional settings and services to configure and automatically maintain the size of the connection pool.

Minimizing the Run-Time Performance Cost Associated with Creating ManagedConnections

Creating ManagedConnections can be expensive depending on the complexity of the Enterprise Information System (EIS) that the ManagedConnection is representing. As a result, you may decide to populate the connection pool with an initial number of ManagedConnections upon startup of WebLogic Server and therefore avoid creating them at run time. You can configure this setting using the initial-capacity element in the `weblogic-ra.xml` descriptor file. The default value for this element is `1` ManagedConnection.

As stated in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2, when an application component requests a connection to an EIS through the resource adapter, WebLogic Server first tries to match the type of connection being requested with any existing and available ManagedConnection in the connection pool. However, if a match is not found, a new ManagedConnection may be created to satisfy the connection request.

WebLogic Server provides a setting to allow a number of additional ManagedConnections to be created automatically when a match is not found. This feature provides you with the flexibility to control connection pool growth over time and the performance hit on the server each time this growth occurs. You can configure this setting using the `capacity-increment` element in the `weblogic-ra.xml` descriptor file. The default value is `1` ManagedConnection.

Since no initiating security principal or request context information is known at WebLogic Server startup, the initial ManagedConnections, configured with `initial-capacity`, are created with a default security context containing a default subject and a client request information of `null`. When additional ManagedConnections—configured with `capacity-increment`—are created, the first ManagedConnection is created with the known initiating principal and client request information of the connection request. The remaining ManagedConnections—up to the `capacity-increment` limit—are created using the same default security context used when creating the initial ManagedConnections.

For more information about configuring the default Resource Principal, refer to Chapter 2, "Security."

Controlling Connection Pool Growth

As more ManagedConnections are created over time, the amount of system resources—such as memory and disk space—that each ManagedConnection consumes increases. Depending on the Enterprise Information System (EIS), this amount may affect the performance of the overall system. To control the effects of ManagedConnections on system resources, WebLogic Server allows you to configure a setting for the allowed maximum number of allocated ManagedConnections.

You configure this setting using the `maximum-capacity` element in the `weblogic-ra.xml` descriptor file. If a new ManagedConnection (or more than one ManagedConnection in the case of `capacity-increment` being greater than one) needs to be created during a connection request, WebLogic Server ensures that no more than the maximum number of allowed ManagedConnections are created. If the maximum number is reached, WebLogic Server attempts to recycle a ManagedConnection from the connection pool. However, if there are no connections to recycle, a warning is logged indicating that the attempt to recycle failed and that the connection request can only be granted for the amount of connections up to the allowed maximum amount. The default value for `maximum-capacity` is `10` ManagedConnections.

Controlling System Resource Usage

Although setting the maximum number of ManagedConnections prevents the server from becoming overloaded by more allocated ManagedConnections than it can handle, it does not control the efficient amount of system resources needed at any given time. WebLogic Server provides a service that monitors the activity of ManagedConnections in the connection pool during the deployment of a resource adapter. If the usage decreases and remains at this level over a period of time, the size of the connection pool is reduced to an efficient amount necessary to adequately satisfy ongoing connection requests.

This system resource usage service is turned on by default. However, to turn off this service, you can set the `shrinking-enabled` element in the `weblogic-ra.xml` descriptor file to `false`. Use the `shrink-period-minutes` element in the `weblogic-ra.xml` descriptor file to set the frequency with which WebLogic Server calculates the need for connection pool size reduction, and if reduction is needed, selectively removes unused ManagedConnections from the pool. The default value of this element is `15` minutes.

Detecting Connection Leaks

As stated in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2, once the application component has completed its use of the EIS connection, it sends a close connection request. At this point, WebLogic Server is responsible for any necessary cleanup and making the connection available for a future connection request. However, if the application component fails to close the connection, the connection pool can exhausted of its available connections, and future connection requests can therefore fail.

WebLogic Server provides a service to prevent the above scenario by automatically closing a ManagedConnection that has exhausted its usage time. You can set the usage time using the `connection-duration-time` element in the `weblogic-ra.xml` descriptor file. You can also use the `connection-cleanup-frequency` element to set the frequency with which WebLogic Server calculates the usage time of the currently used ManagedConnections and closes those that have exceeded their usage time.

To turn off the connection leak detection service, set the `connection-cleanup-frequency` element to `-1`. By default, this service is turned off. The unit used in these element values is seconds.

Monitoring Connection Pools Using the Console

To monitor all connection pool run times for a connector using the BEA WebLogic Server Administrative Console, proceed as follows:

1. Select a connector to monitor in the left pane of the Console.
2. Right-click with your mouse, and select Monitor all Connector Connection Pool Runtimes from the pop-up menu.

   Connection pool run-time information is provided in the right pane for the selected connector.

CHAPTER

5 Configuration

The following sections outline configuration requirements for the WebLogic J2EE Connector Architecture implementation:

- Resource Adapter Developer Tools
- Resource Adapter Developer Tools
- Configuring the ra.xml File
- Configuring the weblogic-ra.xml File
- Configuring the Security Principal Map
- Using the Password Converter Tool
- Configuring the Transaction Level Type

Resource Adapter Developer Tools

BEA provides several tools you can use to help you create and configure resource adapters. These tools are described in this section.

ANT Tasks to Create Skeleton Deployment Descriptors

You can use the WebLogic ANT utilities to create skeleton deployment descriptors. These utilities are Java classes shipped with your WebLogic Server distribution. The ANT task looks at a directory containing a resource adapter creates deployment descriptors based on the files it finds in the resource adapter. Because the ANT utility does not have information about all of the desired configurations and mappings for your resource adapter, the skeleton deployment descriptors the utility creates are incomplete. After the utility creates the skeleton deployment descriptors, you can use a text editor, an XML editor, or the Administration Console to edit the deployment descriptors and complete the configuration of your resource adapter.

For more information on using ANT utilities to create deployment descriptors, see Packaging Resource Adapters.

Resource Adapter Deployment Descriptor Editor

The WebLogic Server Administration Console has an integrated deployment descriptor editor. You must create at least a skeleton ra.xml deployment descriptor before using this integrated editor. For more information, see Appendix A, “weblogic-ra.xml Deployment Descriptor Elements.”

XML Editor

BEA now provides a simple, user-friendly tool from Ensemble for creating and editing XML files. It can validate XML code according to a specified DTD or XML Schema. The XML editor can be used on Windows or Solaris machines and is downloadable from BEA dev2dev at http://dev2dev.bea.com/resourcelibrary/utilitiestools/index.jsp.

Configuring Resource Adapters

This section introduces and discusses how to configure the resource adapter for deployment to WebLogic Server.

Overview of the Resource Adapter

The J2EE Connector Architecture enables both Enterprise Information System (EIS) vendors and third-party application developers to develop resource adapters that can be deployed in any application server supporting the Sun Microsystems J2EE Platform Specification, Version 1.3.

The resource adapter is the central piece of the WebLogic J2EE Connector Architecture; it serves as the J2EE connector between the client component and the EIS. When a resource adapter is deployed in the WebLogic Server environment, it enables the development of robust J2EE Platform applications that can access remote EIS systems. Resource adapters contain the Java components, and if necessary, the native components required to interact with the EIS.

For more information on creating resource adapters, see the Sun Microsystems J2EE Connector Architecture page and the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2. These can be found on the Sun Microsystems Web site at the following respective URLs:

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

http://java.sun.com/j2ee/download.html#connectorspec

Creating and Modifying Resource Adapters: Main Steps

Creating a resource adapter requires creating the classes for the particular resource adapter (ConnectionFactory, Connection, and so on) and the connector-specific deployment descriptors, and then packaging everything up into an `jar` file to be deployed to WebLogic Server.

Creating a New Resource Adapter (.rar)

The following are the main steps for creating a resource adapter (`.rar`):

1. Write the Java code for the various classes required by resource adapter (ConnectionFactory, Connection, and so on) in accordance with the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec).

   When implementing a resource adapter, you must specify classes in the `ra.xml` file. For example:

   - `<managedconnectionfactory-class>com.sun.connector.blackbox.LocalTxManagedConnectionFactory</managedconnectionfactory-class>`
   - `<connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>`
   - `<connectionfactory-impl-class>com.sun.connector.blackbox.JdbcDataSource</connectionfactory-impl-class>`
   - `<connection-interface>java.sql.Connection</connection-interface>`
   - `<connection-impl-class>com.sun.connector.blackbox.JdbcConnection</connection-impl-class>`

2. Compile the Java code for the interfaces and implementation into class files.

   For detailed information about compiling, refer to "Preparing to Compile" in "Developing WebLogic Server Applications."

3. Package the Java classes into a Java archive (`.jar`) file.

4. Create the resource adapter-specific deployment descriptors:

   - `ra.xml` describes the resource adapter-related attributes type and its deployment properties using a standard DTD from Sun Microsystems.
   - `weblogic-ra.xml` adds additional WebLogic Server-specific deployment information.

   For more information, refer to "Configuring the ra.xml File" and "Configuring the weblogic-ra.xml File" on page 5-11.

   Note: If your resource adapter `.rar` does not contain a `weblogic-ra.xml` file, WebLogic Server automatically generates this file for you. For more information, see "Automatic Generation of the weblogic-ra.xml File."

5. Create a resource adapter archive file (`.rar` file).

   a. The first step is to create an empty staging directory.

   b. Place the `.rar` file containing the resource adapter Java classes in the staging directory.

   c. Then, place the deployment descriptors in a subdirectory called `META-INF`.

   d. Next, create the resource adapter archive by executing a `jar` command like the following in the staging directory:

      ```
      jar cvf myRAR.rar *
      ```

      For detailed information about creating the resource adapter archive file, refer to "Packaging Resource Adapters (.rar)" on page 5-10.

6. Deploy the `.rar` resource adapter archive file on WebLogic Server or include it in an enterprise archive (`.ear`) file to be deployed as part of an enterprise application.

   Refer to "Deploying Applications and Components" in "Packaging and Deploying WebLogic Server Applications" for detailed information about deploying components and applications.

Modifying an Existing Resource Adapter (.rar)

The following is an example of how to take an existing resource adapter (`.rar`) and modify it for deployment to WebLogic Server. This involves adding the `weblogic-ra.xml` deployment descriptor and repacking.

1. Create a temporary directory to stage the resource adapter:

   ```
   mkdir c:/stagedir
   ```

2. Copy the resource adapter that you will deploy into the temporary directory:

   ```
   cp blackbox-notx.rar c:/stagedir
   ```

3. Extract the contents of the resource adapter archive:

   ```
   cd c:/stagedir
   
   jar xf blackbox-notx.rar
   ```

   The staging directory should now contain the following:

   - A `jar` file containing Java classes that implement the resource adapter

- A `META-INF` directory containing the files: `Manifest.mf` and `ra.xml`

Execute these commands to see these files:

```
c:/stagedir> ls

        blackbox-notx.jar

        META-INF

c:/stagedir> ls META-INF

        Manifest.mf

        ra.xml
```

4. Create the `weblogic-ra.xml` file. This file is the WebLogic-specific deployment descriptor for resource adapters. In this file, you specify parameters for connection factories, connection pools, and security mappings.

   Note: If your resource adapter `.rar` does not contain a `weblogic-ra.xml` file, WebLogic Server automatically generates this file for you. For more information, see “Automatic Generation of the weblogic-ra.xml File.”

   Refer to “Configuring the weblogic-ra.xml File” on page 5-12 and Appendix A, “weblogic-ra.xml Deployment Descriptor Elements,” for more information on the `weblogic-ra.xml` file.

5. Copy the `weblogic-ra.xml` file into the temporary directory's `META-INF` subdirectory. The `META-INF` directory is located in the temporary directory where you extracted the `.rar` file or in the directory containing a resource adapter in exploded directory format. Use the following command:

```
cp weblogic-ra.xml c:/stagedir/META-INF

c:/stagedir> ls META-INF

        Manifest.mf

        ra.xml

        weblogic-ra.xml
```

6. Create the resource adapter archive:

```
jar cvf blackbox-notx.jar -C c:/stagedir
```

7. Deploy the resource adapter in WebLogic Server. For more information on deploying a resource adapter in WebLogic Server, see Chapter 7, “Resource Adapter Deployment.”

Automatic Generation of the weblogic-ra.xml File

If your resource adapter `.rar` does not contain a `weblogic-ra.xml` file, WebLogic Server automatically generates this file for you. This feature enables you to deploy third-party resource adapters to WebLogic Server without worrying about modifying them for WebLogic Server. You need only modify two default attribute values that WebLogic Server generates in the `weblogic-ra.xml` file: `<connection-factory-name>` and `<jndi-name>`.

- WebLogic Server prepends `<connection-factory-name>` with the default value of `__TMP_CFNAME_`.
- It prepends `<jndi-name>` with the default value of `__TMP_JNDINAME_`.

For instructions on how to change these default values, see “Using the Console Deployment Descriptor Editor to Edit Files” in Appendix A, “weblogic-ra.xml Deployment Descriptor Elements.”

The following is what the generated `weblogic-ra.xml` file looks like before you change the default values:

Listing 5-1 weblogic-ra.xml Default Values

```
<weblogic-connection-factory-dd>

<connection-factory-name>__TMP_CFNAME_.\config\mydomain\applicati
ons\whitebox-notx.rar</connection-factory-name>

<jndi-name>__TMP_JNDINAME_.\config\mydomain\applications\whitebox
-notx.rar</jndi-name>

       <pool-params>

              <initial-capacity>0</initial-capacity>

              <max-capacity>1</max-capacity>

              <capacity-increment>1</capacity-increment>

              <shrinking-enabled>false</shrinking-enabled>
```

```
            <shrink-period-minutes>200</shrink-period-minutes>
        </pool-params>
        <security-principal-map>
        </security-principal-map>
</weblogic-connection-factory-dd>
```

Configuring the ra-link-ref Element

The optional `<ra-link-ref>` element allows you to associate multiple deployed resource adapters with a single deployed resource adapter. In other words, it allows you to link (reuse) resources already configured in a base resource adapter to another resource adapter, modifying only a subset of attributes. The `<ra-link-ref>` element enables you to avoid—where possible—duplicating resources (such as classes, `.jar` files, image files, and so on). Any values defined in the base resource adapter deployment are inherited by the linked resource adapter, unless otherwise specified in the `<ra-link-ref>` element.

If you use the optional `<ra-link-ref>` element, you must provide either *all* or *none* of the values in the `<pool-params>` element. The `<pool-params>` element values are not partially inherited by the linked resource adapter from the base resource adapter.

Do one of the following:

- Assign the `<max-capacity>` element the value of `0` (zero) using the Console Deployment Descriptor Editor. This allows the linked resource adapter to inherit its `<pool-params>` element values from the base resource adapter.
- Assign the `<max-capacity>` element any value other than `0` (zero). The linked resource adapter will inherit no values from the base resource adapter. If you choose this option, you must specify *all* of the `<pool-params>` element values for the linked resource adapter.

For instructions on editing the `weblogic-ra.xml` file, see “Using the Console Deployment Descriptor Editor to Edit Files” in Appendix A, “weblogic-ra.xml Deployment Descriptor Elements.”

Packaging Guidelines

A resource adapter is a WebLogic Server component contained in an `.rar` archive file within the `applications/` directory. The deployment process begins with the `.rar` file or a deployment directory, both of which contain the compiled resource adapter interfaces and implementation classes created by the resource adapter provider. Regardless of whether the compiled classes are stored in an `.rar` file or a deployment directory, they must reside in subdirectories that match their Java package structures.

Resource adapters use a common directory format. This same format is used when a resource adapter is packaged in an exploded directory format as an `.rar` file. A resource adapter is structured as in the following example:

Listing 5-2 Resource Adapter Structure

```
/META-INF/ra.xml

/META-INF/weblogic-ra.xml


/images/ra.jpg


/readme.html

/eis.jar

/utilities.jar

/windows.dll

unix.so
```

Note the following about the files in a resource adapter:

- Deployment descriptors (`ra.xml` and `weblogic-ra.xml`) must be in a subdirectory called `META-INF`.

- The resource adapter can contain multiple `jar` files that contain the Java classes and interfaces used by the resource adapter. (For example, `eis.jar` and `utilities.jar`)
- The resource adapter can contain native libraries required by the resource adapter for interacting with the EIS. (For example, `windows.dll` and `unix.so`)

The resource adapter can include documentation and related files not directly used by the resource adapter. (For example, `readme.html` and `/images/ra.jpg`)

Packaging Resource Adapters (.rar)

You can stage one or more resource adapters in a directory and package them in a `jar` file. Before you package your resource adapters, be sure you read and understand "Resolving Class References Between Components" in "Packaging and Deploying WebLogic Server Applications," which describes how WebLogic Server loads classes.

To stage and package a resource adapter:

1. Create a temporary staging directory.
2. Compile or copy the resource adapter Java classes into the staging directory.
3. Create a `.jar` file to store the resource adapter Java classes. Add this `.jar` file to the top level of the staging directory.
4. Create a `META-INF` subdirectory in the staging directory.
5. Create an `ra.xml` deployment descriptor in the `META-INF` subdirectory and add entries for the resource adapter.

   Note: Refer to the following Sun Microsystems documentation for information on the `ra.xml` document type definition at: http://java.sun.com/dtd/connector_1_0.dtd
6. Create a `weblogic-ra.xml` deployment descriptor in the `META-INF` subdirectory and add entries for the resource adapter.

   Note: Refer to Appendix A, "weblogic-ra.xml Deployment Descriptor Elements," for information on the `weblogic-ra.xml` document type definition.

7. When all of the resource adapter classes and deployment descriptors are set up in the staging directory, you can create the resource adapter JAR file with a `jar` command such as:

   ```
   jar cvf jar-file.rar -C staging-dir.
   ```

   This command creates a `jar` file that you can deploy on a WebLogic Server or package in an application JAR file.

   The `-C` *`staging-dir`* option instructs the `jar` command to change to the `staging-dir` directory so that the directory paths recorded in the JAR file are relative to the directory where you staged the resource adapters.

For instructions on creating a resource adapter and modifying an existing resource adapter for deployment to WebLogic Server, see "Creating and Modifying Resource Adapters: Main Steps" on page 5-3.

Configuring the ra.xml File

If you do not have an `ra.xml` file, you must manually create or edit an existing one to set the necessary deployment properties for the resource adapter. You can use a text editor to edit the properties. For information on creating an `ra.xml` file, refer to the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2: http://java.sun.com/j2ee/download.html#connectorspec

Configuring the weblogic-ra.xml File

The `weblogic-ra.xml` file contains information required for deploying a resource adapter in WebLogic Server. In this file, you specify certain attributes. This functionality is consistent with the equivalent `.xml` extensions for EJBs and Web applications in WebLogic Server, which also add WebLogic-specific deployment descriptors to the deployable archive.

These sections describe how to configure the `weblogic-ra.xml` file to define WebLogic Server-specific content for deployment to WebLogic Server.

Configuring the weblogic-ra.xml File

As is, the basic `.rar` or deployment directory cannot be deployed to WebLogic Server. You must first create and configure WebLogic Server-specific deployment properties in the `weblogic-ra.xml` file, and add that file to the deployment.

The `weblogic-ra.xml` file defines the connection factory, connection pool parameters, security principal mapping parameters, and more. See Appendix A, “weblogic-ra.xml Deployment Descriptor Elements,” for a complete list of properties available in the file.

Configurable weblogic-ra.xml Entities

The `weblogic-ra.xml` file contains information required for deploying a resource adapter in WebLogic Server. In this file, you specify the following attributes:

- Name of the connection factory.
- Descriptive text about the connection factory.
- JNDI name bound to a connection factory.
- Reference to a separately deployed connection factory that contains resource adapter components that can be shared with the current resource adapter.
- Directory where all shared libraries should be copied.
- Connection pool parameters that set the following behavior:
  - Initial number of managed connections WebLogic Server attempts to allocate at deployment time.
  - Maximum number of managed connections WebLogic Server allows to be allocated at any one time.
  - Number of managed connections WebLogic Server attempts to allocate when filling a request for a new connection.
  - Whether WebLogic Server attempts to reclaim unused managed connections to save system resources.

  - The time WebLogic Server waits between attempts to reclaim unused managed connections.
  - The frequency of time to detect and reclaim connections that have exceeded their usage time.
  - The amount of usage time allowed for a connection.

- Values for configuration properties defined in a `<config-entry>` element of the J2EE resource adapter deployment descriptor, `ra.xml`.
- Mapping of security principals for Resource Adapter/EIS sign-on processing. This mapping identifies resource principals to be used when requesting EIS connections for applications that use container-managed security and for EIS connections requested during initial deployment.
- Flag to indicate whether logging is required for the ManagedConnectionFactory or ManagedConnection.
- File to store logging information for the ManagedConnectionFactory or ManagedConnection.

Note: Refer to the `weblogic-ra.xml` DTD in Appendix A, "weblogic-ra.xml Deployment Descriptor Elements," for more information on setting the parameters in `weblogic-ra.xml`. You can also look at the `weblogic-ra.xml` file in the included Simple Black Box resource adapter example provided with the product download.

Note: For information on configuring connection properties in a resource adapter, refer to Chapter 4, "Connection Management."

Configuring the Security Principal Map

To use container-managed sign-on, WebLogic Server must identify a resource principal and then request the connection to the EIS on behalf of the resource principal. In order to make this identification, WebLogic Server looks for a security principal map that you have specified with the `<security-principal-map>` element in the `weblogic-ra.xml` deployment descriptor file.

This map builds associations between WebLogic Server initiating principals (WebLogic Server users with identities defined in the WebLogic Security Realm) and resource principals (users known to the resource adapter / EIS system).

In addition, the `<security-principal-map>` enables you to define a default initiating principal that you can map to an appropriate resource principal when the initiating principal identified at run time is not found in the mapping. You establish the default initiating principal in the `<security-principal-map>` element with an `<initiating-principal>` element that has a value of `*`, for example:

```
<initiating-principal>*</initiating-principal>
```

You must also include a corresponding `<resource-principal>` entry in the `<security-principal-map>` element that specifies a username and password.

The following example shows an association between a WebLogic Server initiating principal and a resource principal.

Listing 5-3 Example <initiating-principal> and <resource-principal> Entry

```
<security-principal-map>
   <map-entry>
       <initiating-principal>*</initiating-principal>
          <resource-principal>
              <resource-username>default</resource-username>
              <resource-password>try</resource-password>
          </resource-principal>
   </map-entry>
</security-principal-map>
```

This default initiating principal mapping is also used at deployment time if the connection pool parameters indicate that WebLogic Server should initialize connections. The absence of a default initiating principal entry or the absence of a `<security-principal-map>` element may prevent WebLogic Server from creating connections using container-managed security.

Using the Password Converter Tool

Because current configuration and packaging requirements for resource adapters in WebLogic Server require manual editing of the `weblogic-ra.xml` file, any new passwords specified in the security-principal-map entries are done in clear-text.

Because BEA understands the importance of protecting security passwords, it provides a converter tool that allows for the encryption of all passwords present in the `weblogic-ra.xml` file. The converter tool is shipped in the standard `weblogic.jar` file.

If you require a resource adapter to have clear-text passwords that do not exist in the WebLogic Server environment, then you must post-process the resultant `weblogic-ra.xml` file by using the converter tool each time a new clear-text password is added.

You must run the provided password converter tool to convert all resource-password values that are in clear text to encrypted password values. This converter tool parses an existing `weblogic-ra.xml` file containing clear-text passwords and creates a new `weblogic-ra.xml` file that contains encrypted passwords. This is the new file that you package in the `.rar` file for deployment to WebLogic Server.

How to Execute

To run the converter tool, execute the following syntax in a DOS command shell:

Listing 5-4 Converter Tool Syntax

```
java weblogic.Connector.ConnectorXMLEncrypt
<input-weblogic-ra.xml> <output-weblogic-ra.xml>
<domain-config-directory-location>
```

Security Hint

A security hint that is specific to the domain used in the encryption / decryption process requires inclusion of the `<domain config directory location>`; the converter tool must be directed to use the specific hint for this domain. The resultant encrypted passwords are specific to this domain. Therefore, the resultant `.rar` with encrypted password values are deployable only on the specified domain.

Configuring the Transaction Level Type

You must specify the transaction level type supported by the resource adapter in the `ra.xml` deployment descriptor file. To specify the transaction support level:

- For No Transaction, add the following entry to the `ra.xml` deployment descriptor file:
  `<transaction-support>NoTransaction</transaction-support>`
- For XA Transaction, add the following entry to the `ra.xml` deployment descriptor file:
  `<transaction-support>XATransaction</transaction-support>`
- For Local Transaction, add the following entry to the `ra.xml` deployment descriptor file:
  `<transaction-support>LocalTransaction</transaction-support>`

For instructions on editing an `.xml` file, see Manually Editing XML Deployment Files and Using the Console Deployment Descriptor Editor to Edit Files in Appendix A, "weblogic-ra.xml Deployment Descriptor Elements."

For more information on specifying the transaction level in the `.rar` configuration, see "Resource Adapter XML DTD" under "Packaging and Deployment" in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec).

CHAPTER

6 Writing J2EE Connector Architecture-Compliant Resource Adapters

The following sections identify the requirements for developing a compliant Resource Adapter, as identified in the J2EE Platform Specification, Version 1.3, Proposed Final Draft 3—http://java.sun.com/j2ee. The following sections correspond to the System Contract requirements identified in this specification:

- Connection Management
- Security Management
- Transaction Management
- Packaging and Deployment

In addition, any Weblogic specific limitations/restrictions are identified.

Note: For instructions on building a resource adapter, see the BEA WebLogic Application Integration documentation at: http://edocs.bea.com/wlintegration/v2_0/applicationintegration/devel/index.htm

Connection Management

The connection management contract requirements for a resource adapter are as follows:

- A resource adapter must provide implementations of the following interfaces:
  - `javax.resource.spi.ManagedConnectionFactory`
  - `javax.resource.spi.ManagedConnection`
  - `javax.resource.spi.ManagedConnectionMetaData`
- The `ManagedConnection` implementation provided by a resource adapter must use the following interface and classes to provide support to an application server for connection management (and transaction management, as explained later):
  - `javax.resource.spi.ConnectionEvent`
  - `javax.resource.spi.ConnectionEventListener`

  To support non-managed environments, a resource adapter is not required to use the above two interfaces to drive its internal object interactions.
- A resource adapter is required to provide support for basic error logging and tracing by implementing the following methods:
  - `ManagedConnectionFactory.set/getLogWriter`
  - `ManagedConnnection.set/getLogWriter`
- A resource adapter is required to provide a default implementation of the `javax.resource.spi.ConnectionManager` interface. The implementation class comes into play when a resource adapter is used in a non-managed two-tier application scenario. In an application server-managed environment, the resource adapter should not use the default `ConnectionManager` implementation class.

  A default implementation of `ConnectionManager` enables the resource adapter to provide services specific to itself. These services can include connection pooling, error logging and tracing, and security management. The default `ConnectionManager` delegates to the `ManagedConnectionFactory` the creation of physical connections to the underlying EIS.
- In a managed environment, a resource adapter is not allowed to support its own internal connection pooling. In this case, the application server is responsible for

connection pooling. However, a resource adapter may multiplex connections (one or more `ConnectionManager` instances per physical connection) over a single physical pipe transparent to the application server and components.

In a non-managed two-tier application scenario, a resource adapter is allowed to support connection pooling internal to the resource adapter.

Security Management

The security management contract requirements for a resource adapter are as follows:

- The resource adapter is required to support the security contract by implementing the method `ManagedConnectionFactory.createManagedConnection`.
- The resource adapter is not required to support re-authentication as part of its `ManagedConnection.getConnection` method implementation.
- The resource adapter is required to specify its support for the security contract as part of its deployment descriptor. The relevant deployment descriptor elements are: `authentication-mechanism`, `authentication-mechanism-type`, `reauthentication-support` and `credential-interface.` Refer to section 10.6, “Resource Adapter XML DTD,” of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec).

Transaction Management

This section outlines the transaction management contract requirements for a resource adapter. A resource adapter can be classified based on the level of transaction support, as follows:

- Level `NoTransaction`—The resource adapter supports neither resource manager local nor JTA transactions. It implements neither `XAResource` nor `LocalTransaction` interfaces.

- Level `LocalTransaction`—The resource adapter supports resource manager local transactions by implementing the `LocalTransaction` interface. The local transaction management contract is specified in section 6.7 of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec).
- Level `XATransaction`—The resource adapter supports both resource manager local and JTA transactions by implementing `LocalTransaction` and `XAResource` interfaces respectively. The requirements for support XAResource-based contract are specified in section 6.6 of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec).

Note: Other levels of support (includes any transaction optimizations supported by an underlying resource manager) are outside the scope of the Connector Architecture.

The above levels reflect the major steps of transaction support that a resource adapter needs to make to allow external transaction coordination. Depending on its transaction capabilities and requirements of its underlying EIS, a resource adapter can choose to support any one of the above transaction support levels.

Packaging and Deployment

The following sections discuss packaging and deployment requirements for resource adapters.

Restrictions

At present, the WebLogic J2EE Connector Architecture supports only serializable ConnectionFactory implementations. (Referenceable-only ConnectionFactory implementations are not supported.)

In addition, the WebLogic J2EE Connector Architecture does not support the `javax.resource.spi.security.GenericCredential` credential-interface or the Kerbv5 `authentication-mechanism-type`.

Specification of either of these values for the `<authentication-mechanism>` in the `ra.xml` file for the resource adapter being deployed will result in a failed deployment.

Packaging

The file format for a packaged resource adapter module defines the contract between a resource adapter provider and deployer. A packaged resource adapter includes the following elements:

- Java classes and interfaces that are required for the implementation of both the Connector Architecture contracts and the functionality of the resource adapter
- Utility Java classes for the resource adapter
- Platform-dependent native libraries required by the resource adapter
- Help files and documentation
- Descriptive meta information that ties the above elements together

Deployment

For information on packaging requirements, refer to section 10.3.1, "Resource Adapter Provider," and section 10.5.1, "Responsibilities," of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 (http://java.sun.com/j2ee/download.html#connectorspec). Section 10.3.1 discusses deployment requirements, while section 10.5.1 discusses how to support JNDI Configuration and Lookup.

CHAPTER

7 Resource Adapter Deployment

The following sections explain how to deploy, undeploy, and update the deployment of configured resource adapters to WebLogic Server:

- Resource Adapter Deployment Overview
- Using the Administration Console
- Using the Applications Directory
- Using weblogic.deploy
- Including a Resource Adapter in an Enterprise Application (.ear file)

Resource Adapter Deployment Overview

Deployment of a resource adapter is similar to deployment of Web Applications, EJBs, and Enterprise Applications. Like these deployment units, you can deploy a resource adapter in an exploded directory format or as an archive file.

Deployment Options

You can deploy a resource adapter:

- Dynamically using command line or through the Administration Console
- Automatically while WebLogic Server is running by copying the archive file or exploded directory into the `applications` directory of a WebLogic Server domain
- As part of an Enterprise Application, which is deployed in an archive file called an `.ear`

Deployment Descriptor

Also similar to Web Applications, EJBs, and Enterprise Applications, resource adapters use two deployment descriptors to define their operational parameters. The deployment descriptor `ra.xml` is defined by Sun Microsystems in the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2. The `weblogic-ra.xml` deployment descriptor is specific to WebLogic Server and defines operational parameters unique to WebLogic Server. For more information about the `weblogic-ra.xml` deployment descriptor, refer to Appendix A, “weblogic-ra.xml Deployment Descriptor Elements.”

Resource Adapter Deployment Names

When you deploy a resource adapter `.rar` file or deployment directory, you must specify a name for the deployment unit, for example, `myResourceAdapter`. This name provides a shorthand reference to the resource adapter deployment that you can later use to undeploy or update the resource adapter.

When you deploy a resource adapter, WebLogic Server implicitly assigns a deployment name that matches the path and filename of the `.rar` file or deployment directory. You can use this assigned name to undeploy or update the resource adapter after the server has started.

The resource adapter deployment name remains active in WebLogic Server until the server is rebooted. Undeploying a resource adapter does not remove the associated deployment name, because you may later re-use that name to deploy the resource adapter.

Using the Administration Console

This section discusses resource adapter deployment tasks using the Administration Console.

Deploying Resource Adapters Using the Administration Console

To deploy a resource adapter using the WebLogic Server Administration Console:

1. Start WebLogic Server.
2. Open the Administration Console.
3. Open the Domain you will be working in.

4. Under Deployments, select Connectors in the left panel. The Connector Deployments table displays in the right pane showing all the deployed Connectors (Resource Adapters).
5. Select Configure a new Connector Component...
6. Enter the following information:
   - Name—modify the default name of the connector component as needed.
   - URI Path—enter the full path of the resource adapter `.rar` file or a directory containing the resource adapter exploded directory format. For example: `c:\myaps\components\myResourceAdapter.rar`
   - Deployed—indicate whether the Resource Adapter `.rar` file should be deployed when created.
7. Select the Create button.
8. Note that the new resource adapter now appears in the Deployments table in the right pane.

Viewing Deployed Resource Adapters Using the Administration Console

To view a deployed resource adapter in the Administration Console:

1. In the Administration Console under Deployments, select Connectors (Resource Adapters) in the left panel.
2. View a list of deployed Connectors in the Connector Deployments table in the right pane.

Undeploying Deployed Resource Adapters Using the Administration Console

To undeploy a deployed resource adapter from the WebLogic Server Administration Console:

1. In the Administration Console under Deployments, select Connectors (Resource Adapters) in the left panel.
2. In the Connector Deployments table, select the connector to undeploy.
3. Under the Configuration tab, de-select the Deployed checkbox.
4. Click Apply.

Undeploying a resource adapter does not remove the resource adapter name from WebLogic Server. The resource adapter remains undeployed for the duration of the Server session, as long as you do not change it once it has been undeployed. You cannot re-use the deployment name with the deploy argument until you reboot the server. You can re-use the deployment name to update the deployment, as described in the following section.

Updating Deployed Resource Adapters Using the Administration Console

When you update the contents of the resource adapter `.rar` file or deployment directory that has been deployed to WebLogic Server, those updates are not reflected in WebLogic Server until:

- You reboot the server (if the `.rar` or directory is to be automatically deployed)

  or

- You update the resource adapter deployment using the WebLogic Server Administration Console

From the WebLogic Server Administration Console:

1. In the Administration Console under Deployments, select Connectors (Resource Adapters) in the left panel.
2. In the Connector Deployments table, select the connector to update.
3. Update the Connector Name and Deployed status as needed.
4. Click Apply.

Using the Applications Directory

You can deploy a resource adapter automatically while WebLogic Server is running. The `/applications` directory is monitored during runtime of the WebLogic Server and detects if a new `.rar` is added (causing deployment) or if an existing `.rar` is removed (causing undeployment).

To deploy a resource adapter using the `applications` directory:

1. Copy the `.rar` archive or the exploded directory containing a resource adapter to the `applications` directory of a domain.

   For example, after copying a resource adapter called `myResourceAdapter` in exploded format, your WebLogic Server installation looks like this (not all the files for a resource adapter are shown here):

Listing 7-1   myResourceAdapter

```
\---beaHome
        \---wlserver6.x
                \---config
                        \---mydomain
                                \---applications
                                        \---myResourceAdapter
                                                eis.jar
                                                readme.html
                                                unix.so
                                                utilities.jar
                                                windows.dll
                                                \---META-INF
                                                        ra.xml
                                                        weblogic-ra.xml
```

After copying a `.rar` file, your WebLogic Server installation looks like this:

Listing 7-2   rar File

```
        \---beaHome
                \---wlserver6.x
                        \---config
                                \---mydomain
                                        \---applications
                                                myResourceAdapter.rar
```

2. Start WebLogic Server. When you boot WebLogic Server, it automatically attempts to deploy the specified resource adapter `.rar` file or deployment directory.
3. Launch the Administration Console.
4. In the right pane, click Resource Adapter Deployments.
5. Verify that the resource adapter is listed and that the Deployed box is selected.

Using weblogic.deploy

To deploy a resource adapter `.rar` file or deployment directory that has not been deployed to WebLogic Server, use the following command:

```
% java weblogic.deploy -port port_number -host host_name

deploy password name source
```

where:

- *`name`* is the string containing the name you want to assign to this resource adapter deployment unit.
- *`password`* is the password for the WebLogic Server system account.
- *`source`* is the full path and filename of the resource adapter `.rar` file you want to deploy, or the full path of the resource adapter deployment directory.

Viewing Deployed Resource Adapters Using weblogic.deploy

To view resource adapters that are deployed on a local WebLogic Server, from the command line enter the following:

```
% java weblogic.deploy list password
```

where *`password`* is the password for the WebLogic Server System account.

To list deployed resource adapters on a remote server, from the command line enter the following:

```
% java weblogic.deploy -port port_number -host host_name
list password
```

Undeploying Deployed Resource Adapters Using weblogic.deploy

Undeploying a resource adapter does not remove the deployment name associated with the resource adapter `.rar` file or deployment directory. The deployment name remains in the server to allow for later updates of the resource adapter.

To undeploy a deployed resource adapter from the command line, you need only reference the assigned deployment unit name, as in:

```
%java weblogic.deploy -port 7001 -host localhost undeploy
password myResourceAdapter
```

Undeploying a resource adapter does not remove the resource adapter name from WebLogic Server. The resource adapter will remain undeployed for the duration of the Server session, as long as you do not change it once it has been undeployed. You cannot re-use the deployment name with the deploy argument until you reboot the server. You can re-use the deployment name to update the deployment, as described in “Updating Deployed Resource Adapters Using weblogic.deploy” on page 7-9.

Updating Deployed Resource Adapters Using weblogic.deploy

When you update the contents of the resource adapter `.rar` file or deployment directory that has been deployed to WebLogic Server, those updates are not reflected in WebLogic Server until:

- You reboot the server (if the `.rar` or directory is to be automatically deployed) or

- You update the resource adapter deployment using the WebLogic Server Administration Console

To update or redeploy the resource adapter from the command line, use the `update` argument and specify the active resource adapter deployment name:

```
%java weblogic.deploy -port 7001 -host localhost update
password myResourceAdapter
```

Including a Resource Adapter in an Enterprise Application (.ear file)

As part of the J2EE Platform Specification, Version 1.3, Proposed Final Draft 3, it is now possible to include a resource adapter archive (`.rar`) file inside an Enterprise Application archive (`.ear`) and then deploy the application in WebLogic Server.

To deploy an Enterprise Application that contains a resource adapter archive:

1. Place the `.rar` file inside the `.ear` archive just as you would a `.war` or `.jar` archive.
2. Create a valid `application.xml` and place it in the `META-INF` directory of the `.ear` archive.

   Note the following when creating an `application.xml`:

   The application deployment descriptor must contain the new `<connector>` element to identify the resource adapter archive within the `.ear` archive. For example:

   ```
   <connector>RevisedBlackBoxNoTx.rar</connector>
   ```

   Because the `<connector>` element is a new addition to the J2EE Platform Specification, Version 1.3, the `application.xml` file must contain the following `DOCTYPE` entry to identify it as a J2EE Platform Specification, Version 1.3 deployment descriptor.

Listing 7-3 DOCTYPE Entry

```
<!DOCTYPE application PUBLIC ‘-//Sun Microsystems, Inc.//DTD
J2EE Application 1.3//EN’
       ‘http://java.sun.com/dtd/application_1_3.dtd’>
```

If you do not use this `DOCTYPE` entry, the resource adapter will not be deployed.

The following listing is an example of an `application.xml` file.

Listing 7-4 application.xml File

```
<application>
       <display-name> ConnectorSampleearApp </display-name>
       <module>
              <connector>RevisedBlackBoxNoTx.rar</connector>
       </module>
       <module>
              <ejb>ejb_basic_beanManaged.jar</ejb>
       </module>
</application>
```

3. Deploy the Enterprise Application in WebLogic Server.

   For general information about deployment of Enterprise Applications, see “Enterprise Applications” in “Understanding WebLogic Server Applications.”

CHAPTER

8 Client Considerations

The following sections discuss WebLogic J2EE Connector Architecture client considerations:

- Common Client Interface (CCI)
- ConnectionFactory and Connection
- Obtaining the ConnectionFactory (Client-JNDI Interaction)

Common Client Interface (CCI)

The client API used by application components for EIS access can be defined as follows:

- The standard common client interface (CCI) discussed in chapter 9, “Common Client Interface,” of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 at: http://java.sun.com/j2ee/download.html#connectorspec.
- A client API specific to the type of a resource adapter and its underlying EIS. An example of such EIS-specific client APIs is JDBC for relational databases.

The CCI is a common client API for accessing EISes. The CCI is targeted towards Enterprise Application Integration (EAI) and enterprise tool vendors.

The J2EE Connector Architecture defines a Common Client Interface (CCI) for EIS access. The CCI defines a standard client API for application components that enables application components and EAI frameworks to drive interactions across heterogeneous EISes.

ConnectionFactory and Connection

A connection factory is a public interface that enables connection to an EIS instance; a ConnectionFactory interface is provided by a resource adapter. An application looks up a ConnectionFactory instance in the JNDI namespace and uses it to obtain EIS connections.

One goal of the J2EE Connector Architecture is to support a consistent application programming model across both CCI and EIS-specific client APIs. This model is achieved through use of a design pattern—specified as an interface template—for both the ConnectionFactory and Connection interfaces.

For more information on this design pattern, see section 5.5.1, “ConnectionFactory and Connection” of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 at: http://java.sun.com/j2ee/download.html#connectorspec

Obtaining the ConnectionFactory (Client-JNDI Interaction)

This section discusses how a connection to an EIS instance is obtained from a ConnectionFactory. For further information, refer to section 5.4.1, “Managed Application Scenario,” of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 at: http://java.sun.com/j2ee/download.html#connectorspec

Obtaining a Connection in a Managed Application

The following tasks are performed when a managed application obtains a connection to an EIS instance from a ConnectionFactory, as specified in the `res-type` variable:

1. The application assembler or component provider specifies the connection factory requirements for an application component by using a deployment descriptor mechanism. For example:
   - `res-ref-name: eis/myEIS`
   - `res-type: javax.resource.cci.ConnectionFactory`
   - `res-auth: Application` or `Container`
2. The person deploying the resource adapter sets the configuration information for the resource adapter.
3. The application server uses a configured resource adapter to create physical connections to the underlying EIS. Refer to Chapter 10 of the J2EE Connector Specification, Version 1.0, Proposed Final Draft 2 for more information on packaging and deployment of resource adapters at: http://java.sun.com/j2ee/download.html#connectorspec
4. The application component looks up a connection factory instance in the component’s environment by using the JNDI interface.

Listing 8-1 JNDI Lookup

```
//obtain the initial JNDI Naming context
Context initctc = new InitialContext();


// perform JNDI lookup to obtain the connection factory
javax.resource.cci.ConnectionFactory cxf =
      (javax.resource.cci.ConnectionFactory)
            initctx.lookup(“java:comp/env/eis/MyEIS”);
```

The JNDI name passed in the method `NamingContext.lookup` is the same as that specified in the `res-ref-name` element of the deployment descriptor. The JNDI lookup results in a connection factory instance of type `java.resource.cci.ConnectionFactory` as specified in the `res-type` element.

5. The application component invokes the `getConnection` method on the connection factory to obtain an EIS connection. The returned connection instance represents an application level handle to an underlying physical connection. An application component obtains multiple connections by calling the method `getConnection` on the connection factory multiple times.

   ```
   javax.resource.cci.Connection cx = cxf.getConnection();
   ```

6. The application component uses the returned connection to access the underlying EIS.

7. After the component finishes with the connection, it closes the connection using the close method on the `Connection` interface.

   ```
   cx.close();
   ```

8. If an application component fails to close an allocated connection after its use, that connection is considered an unused connection. The application server manages the cleanup of unused connections. When a container terminates a component instance, the container cleans up all connections used by that component instance.

Obtaining a Connection in a Non-Managed Application

In a non-managed application scenario, the application developer must follow a similar programming model to that of a managed application. Non-management involves lookup of a connection factory instance, obtaining an EIS connection, using the connection for EIS access, and finally closing the connection.

The following tasks are performed when a non-managed application obtains a connection to an EIS instance from a ConnectionFactory:

1. The application client calls a method on the `javax.resource.cci.ConnectionFactory` instance (returned from the JNDI lookup) to get a connection to the underlying EIS instance.
2. The `ConnectionFactory` instance delegates the connection request from the application to the default `ConnectionManager` instance. The resource adapter provides the default `ConnectionManager` implementation.
3. The `ConnectionManager` instance creates a new physical connection to the underlying EIS instance by calling the `ManagedConnectionFactory.createManagedConnection` method.
4. The `ManagedConnectionFactory` instance handles the createManagedConnection method by creating a new physical connection to the underlying EIS, represented by a `ManagedConnection` instance. The `ManagedConnectionFactory` uses the security information (passed as a Subject instance), any `ConnectionRequestInfo`, and its configured set of properties (such as port number, server name) to create a new `ManagedConnection` instance.
5. The `ConnectionManager` instance calls the `ManagedConnection.getConnection` method to get an application-level connection handle. Calling the `getConnection` method does not necessarily create a new physical connection to the EIS instance. Calling `getConnection` produces a temporary handle that is used by an application to access the underlying physical connection. The actual underlying physical connection is represented by a `ManagedConnection` instance.
6. The `ConnectionManager` instance returns the connection handle to the `ConnectionFactory` instance, which then returns the connection to the application that initiated the connection request.

A weblogic-ra.xml Deployment Descriptor Elements

The following sections provide a complete reference for the WebLogic Server 6.1 specific XML deployment properties used in the WebLogic Server resource adapter archive and an explanation of how to edit the XML deployment file manually. Use these sections if you need to refer to the deployment descriptor used for resource adapters.

Note: If your resource adapter `.rar` does not contain a `weblogic-ra.xml` file, WebLogic Server automatically generates this file for you. For more information, see "Automatic Generation of the weblogic-ra.xml File" in Chapter 5, "Configuration."

- Manually Editing XML Deployment Files
- Using the Console Deployment Descriptor Editor to Edit Files
- weblogic-ra.xml DTD
- weblogic-ra. xml Element Hierarchy Diagram
- weblogic-ra.xml Element Descriptions

Manually Editing XML Deployment Files

To define or make changes to the XML deployment descriptors used in the WebLogic Server resource adapter archive, you must manually define or edit the XML elements in the `weblogic-ra.xml` file.

Basic Conventions

To manually edit XML elements:

- Make sure that you use an ASCII text editor that does not reformat the XML or insert additional characters that could invalidate the file.
- Use the correct case for file and directory names, even if your operating system ignores the case.
- To use the default value for an optional element, you can either omit the entire element definition or specify a blank value. For example:

```
<max-config-property></max-config-property>
```

DOCTYPE Header Information

When editing or creating XML deployment files, it is critical to include the correct `DOCTYPE` header for each deployment file. In particular, using an incorrect `PUBLIC` element within the `DOCTYPE` header can result in parser errors that may be difficult to diagnose.

The header refers to the location and version of the Document Type Definition (DTD) file for the deployment descriptor. Although this header references an external URL at `java.sun.com`, WebLogic Server contains its own copy of the DTD file, so your host server need not have access to the Internet. However, you must still include this `<!DOCTYPE...>` element in your `ra.xml` file, and have it reference the external URL because the version of the DTD contained in this element is used to identify the version of this deployment descriptor.

The entire `DOCTYPE` headers for the `ra.xml` and `weblogic-ra.xml` files are as follows:

XML File	DOCTYPE header
`ra.xml`	`<!DOCTYPE connector PUBLIC '-//Sun Microsystems, Inc.//DTD Connector 1.0//EN' 'http://java.sun.com/dtd/connector_1_0.dtd'>`
`weblogic-ra.xml`	`<!DOCTYPE connector PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0 Connector//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic600-ra.dtd'`

XML files with incorrect header information may yield error messages similar to the following, when used with a utility that parses the XML (such as `ejbc`):

```
SAXException: This document may not have the identifier
'identifier_name'
```

`identifier_name` generally includes the invalid text from the `PUBLIC` element.

Document Type Definitions (DTDs) for Validation

The contents and arrangement of elements in your XML files must conform to the Document Type Definition (DTD) for each file you use. WebLogic Server utilities ignore the DTDs embedded within the `DOCTYPE` header of XML deployment files, and instead use the DTD locations that were installed along with the server. However, the DOCTYPE header information must include a valid URL syntax in order to avoid parser errors.

The following links provide the public DTD locations for XML deployment files used with WebLogic Server:

- `connector_1_0.dtd` contains the DTD for the standard `ra.xml` deployment file, required for all resource adapters. This DTD is maintained as part of the J2EE Connector Specification, Version 1.0; refer to this specification for information about the elements used in the `connector_1_0.dtd` (http://java.sun.com/j2ee/download.html#connectorspec).

- `weblogic-ra.dtd` contains the DTD used for creating `weblogic-ra.xml`, which defines resource adapter properties used for deployment to WebLogic Server. This file is located at `http://www.bea.com/servers/wls600/dtd/weblogic600-ra.dtd`

Note: Most browsers do not display the contents of files having the `.dtd` extension. To view the DTD file contents in your browser, save the links as text files and view them with a text editor.

Using the Console Deployment Descriptor Editor to Edit Files

This section describes the procedure for editing the following resource adapter deployment descriptors using the Administration Console Deployment Descriptor Editor:

- `ra.xml`
- `weblogic-ra.xml`

For detailed information about the elements in the resource adapter deployment descriptors, refer to Programming the WebLogic J2EE Connector Architecture.

To edit the resource adapter deployment descriptors, follow these steps:

1. Invoke the Administration Console in your browser using the following URL:

   http://host:port/console

   where host refers to the name of the computer upon which WebLogic Server is running and port refers to the port number to which it is listening.

2. Click to expand the Deployments node in the left pane.

3. Click to expand the Connectors node under the Deployments node.

4. Right-click the name of the resource adapter whose deployment descriptors you want to edit and choose Edit Connector Descriptor from the drop-down menu.

The Administration Console window appears in a new browser. The left pane contains a tree structure that lists all the elements in the two resource adapter deployment descriptors and the right pane contains a form for the descriptive elements of the `ra.xml` file.

5. To edit, delete, or add elements in the resource adapter deployment descriptors, click to expand the node in the left pane that corresponds to the deployment descriptor file you want to edit, as described in the following list:
   - The RA node contains the elements of the `ra.xml` deployment descriptor.
   - The WebLogic RA node contains the elements of the `weblogic-ra.xml` deployment descriptor.
6. To edit an existing element in one of the resource adapter deployment descriptors, follow these steps:
   a. Navigate the tree in the left pane, clicking on parent elements until you find the element you want to edit.
   b. Click the element. A form appears in the right pane that lists either its attributes or sub-elements.
   c. Edit the text in the form in the right pane.
   d. Click Apply.
7. To add a new element to one of the resource adapter deployment descriptors, follow these steps:
   a. Navigate the tree in the left pane, clicking on parent elements until you find the name of the element you want to create.
   b. Right-click the element and chose Configure a New Element from the drop-down menu.
   c. Enter the element information in the form that appears in the right pane.
   d. Click Create.
8. To delete an existing element from one of the resource adapter deployment descriptors, follow these steps:
   a. Navigate the tree in the left pane, clicking on parent elements until you find the name of the element you want to delete.

   b. Right-click the element and chose Delete Element from the drop-down menu.

   c. Click Yes to confirm that you want to delete the element.

9. Once you have made all your changes to the resource adapter deployment descriptors, click the root element of the tree in the left pane. The root element is the either the name of the resource adapter `*.rar` archive file or the display name of the resource adapter.

10. Click Validate if you want to ensure that the entries in the resource adapter deployment descriptors are valid.

11. Click Persist to write your edits of the deployment descriptor files to disk in addition to WebLogic Server's memory.

weblogic-ra.xml DTD

Listing A-1 weblogic-ra.xml DTD

```
DTD for weblogic-ra.xml

<!--

XML DTD for Weblogic Specific Resource Adapter deployment
descriptor 1.0

-->

<!--

This DTD defines the Weblogic specific deployment information for
defining a deployable Resource Adapter Connection Factory. It
provides for complete specification of all configurable Connection
Factory parameters including Connection Pool parameters, Security
parameters for Resource Principal Mapping and the ability to define
values for configuration parameters which exist in the ra.xml
deployment descriptor.

-->

Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.

<!--
```

```
The weblogic-connection-factory-dd element is the root element of
the Weblogic specific deployment descriptor for the deployed
resource adapter.

-->

<!ELEMENT weblogic-connection-factory-dd (connection-factory-name,
description?, jndi-name, ra-link-ref?, native-libdir?,
pool-params?, logging-enabled?, log-filename?,
map-config-property*, security-principal-map?)>

<!--

The connection-factory-name element defines that logical name that
will be associated with this specific deployment of the Resource
Adapter and its corresponding Connection Factory.

The value of connection-factory-name can be used in other deployed
Resource Adapters via the ra-link-ref element. This will allow
multiple deployed Connection Factories to utilize a common deployed
Resource Adapter, as well as share configuration specifications.

This is a required element.

-->

<!ELEMENT connection-factory-name (#PCDATA)>

<!--

The description element is used to provide text describing the
parent element. The description element should include any
information that the deployer wants to describe about the deployed
Connection Factory.

This is an optional element.

-->

<!ELEMENT description (#PCDATA)>

<!--

The jndi-name element defines the name that will be used to bind
the Connection Factory Object into the Weblogic JNDI Namespace.
Client EJBs and Servlets will use this same JNDI in their defined
Reference Descriptor elements of the weblogic specific deployment
descriptors.

This is a required element.

-->
```

```
<!ELEMENT jndi-name (#PCDATA)>

<!--

The ra-link-ref element allows for the logical association of
multiple deployed Connection Factories with a single deployed
Resource Adapter. The specification of the optional ra-link-ref
element with a value identifying a separately deployed Connection
Factory results in this newly deployed Connection Factory sharing
the Resource Adapter, which had been deployed with the referenced
Connection Factory.

In addition, any values defined in the referred Connection Factories
deployment will be inherited by this newly deployed Connection
Factory unless specified.

This is an optional element.

-->

<!ELEMENT ra-link-ref (#PCDATA)>

<!--

The native-libdir element identifies the directory location to be
used for all native libraries present in this resource adapter
deployment. As part of deployment processing, all encountered
native libraries will be copied to the location specified.

It is the responsibility of the Administrator to perform the
necessary platform actions such that these libraries will be found
during Weblogic Server runtime.

This is a required element IF native libraries are present.

-->

<!ELEMENT native-libdir (#PCDATA)>

<!--

The pool-params element is the root element for providing Connection
Pool specific parameters for this Connection Factory.

Weblogic will use these specifications in controlling the behavior
of the maintained pool of Managed Connections.

This is an optional element. Failure to specify this element or any
of its specific element items will result in default values being
assigned. Refer to the description of each individual element for
the designated default value.
```

```
-->

<!ELEMENT pool-params (initial-capacity?, max-capacity?,
capacity-increment?, shrinking-enabled?, shrink-period-minutes?,
connection-cleanup-frequency?, connection-duration-time?)>

<!--

The initial-capacity element identifies the initial number of
managed connections, which the Weblogic Server will attempt to
obtain during deployment.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.

Default Value: 1

-->

<!ELEMENT initial-capacity (#PCDATA)>

<!--

The max-capacity element identifies the maximum number of managed
connections, which the Weblogic Server will allow. Requests for
newly allocated managed connections beyond this limit will result
in a ResourceAllocationException being returned to the caller.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.

Default Value: 10

-->

<!ELEMENT max-capacity (#PCDATA)>

<!--

The capacity-increment element identifies the number of additional
managed connections, which the Weblogic Server will attempt to
obtain during resizing of the maintained connection pool.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.

Default Value: 1
```

```
-->

<!ELEMENT capacity-increment (#PCDATA)>

<!--

The shrinking-enabled element indicates whether or not the
Connection Pool should have unused Managed Connections reclaimed as
a means to control system resources.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.

Value Range:    true|false

Default Value: true

-->

<!ELEMENT shrinking-enabled (#PCDATA)>

<!--

The shrink-period-minutes element identifies the amount of time the
Connection Pool Management will wait between attempts to reclaim
unused Managed Connections.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.

Default Value: 15

-->

<!ELEMENT shrink-period-minutes (#PCDATA)>

<!--

The connection-cleanup-frequency element identifies the amount of
time (in seconds) the Connection Pool Management will wait between
attempts to destroy Connection handles which have exceeded their
usage duration. This element, used in conjunction with
connection-duration-time, prevents connection leaks when an
Application may have not closed a connection after completing usage.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.
```

```
Default Value:-1

-->

<!ELEMENT connection-cleanup-frequency (#PCDATA)>

<!--

The connection-duration-time element identifies the amount of time
(in seconds) a Connection handle can be active. This element, used
in conjunction with connection-cleanup-frequency, prevents leaks
when an Application may have not closed a connection after
completing usage.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.

Default Value:-1

-->

<!ELEMENT connection-duration-time (#PCDATA)>

<!--

The logging-enabled element indicates whether or not the log writer
is set for either the ManagedConnectionFactory or
ManagedConnection. If this element is set to true, output generated
from either the ManagedConnectionFactory or ManagedConnection will
be sent to the file specified by the log-filename element.

This is an optional element.

Failure to specify this value will result in Weblogic using its
defined default value.

Value Range:    true|false

Default Value: false

-->

<!ELEMENT logging-enabled (#PCDATA)>

<!--

The log-filename element specifies the name of the log file which
output generated from either the ManagedConnectionFactory or a
ManagedConnection are sent.

The full address of the filename is required.
```

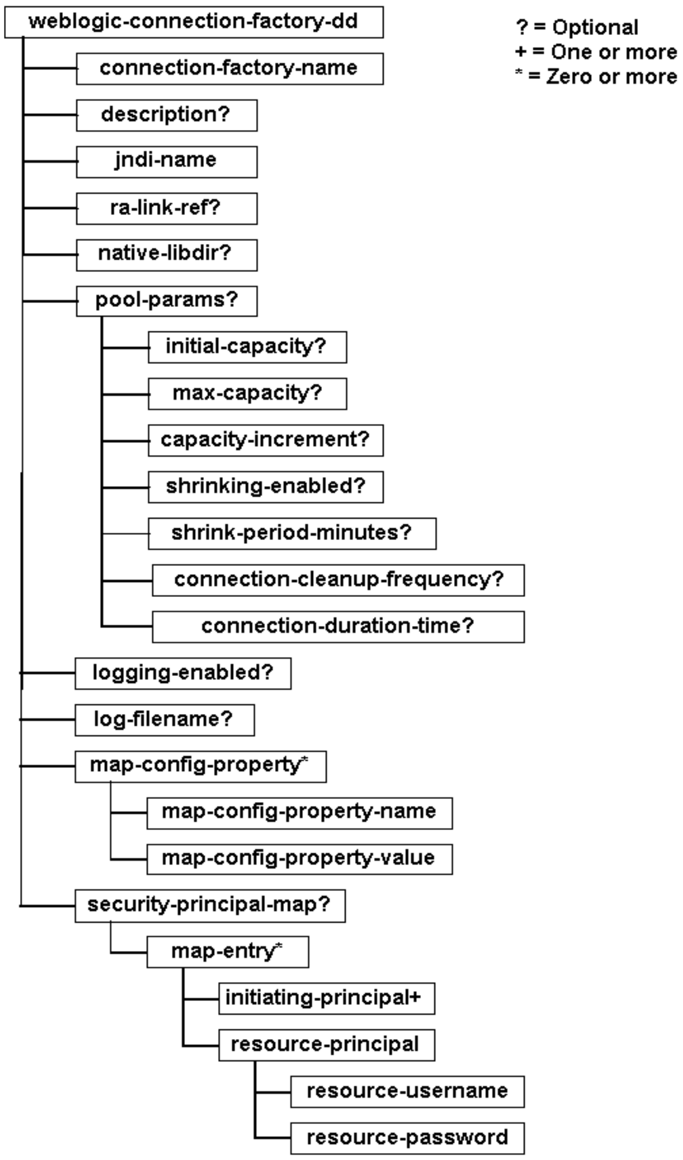
```
This is an optional element.

-->

<!ELEMENT log-filename (#PCDATA)>

<!--

Each map-config-property element identifies a configuration
property name and value that corresponds to an ra.xml config-entry
element with the corresponding config-property-name.



At deployment time, all values present in a map-config-property
specification will be set on the ManagedConnectionFactory.

Values specified via a map-config-property will supersede any
default value that may have been specified in the corresponding
ra.xml config-entry element.

This is an optional element.

-->

<!ELEMENT map-config-property (map-config-property-name,
map-config-property-value)>

<!ELEMENT map-config-property-name (#PCDATA)>

<!ELEMENT map-config-property-value (#PCDATA)>

<!--

Each security-principal-map element provides a mechanism to define
appropriate Resource Principal values for Resource Adapter/EIS
authorization processing, based upon the known Weblogic Runtime
Initiating Principal.

This map allows for the specification of a defined set of Initiating
Principals and the corresponding Resource Principal's Username and
Password that should be used when allocating Managed Connections
and Connection Handles.

A default Resource Principal can be defined for the Connection
Factory via the map. By specifying an initiating-principal value of
'*' and a corresponding resource-principal, the defined
resource-principal will be utilized whenever the current identity
is NOT matched elsewhere in the map.
```

```
This is an optional element, however, it must be specified in some
form if Container Managed Sign-on is supported by the Resource
Adapter and used by ANY client.

In addition, the deployment-time population of the Connection Pool
with Managed Connections will be attempted using the defined
'default' resource principal if one is specified.

-->

<!ELEMENT security-principal-map (map-entry*)>

<!ELEMENT map-entry (initiating-principal+, resource-principal)>

<!ELEMENT initiating-principal (#PCDATA)>

<!ELEMENT resource-principal (resource-username,
resource-password)>

<!ELEMENT resource-username (#PCDATA)>

<!ELEMENT resource-password (#PCDATA)>
```

weblogic-ra. xml Element Hierarchy Diagram

The following diagram summarizes the structure of the `weblogic-ra.xml` deployment descriptor.

Listing A-2 weblogic-ra.xml Element Hierarchy



weblogic-ra.xml Element Descriptions

The following sections describe each of the elements that can be defined in the `weblogic-ra.xml` file:

- *`weblogic-connection-factory-dd`* (required)—the root element of the Weblogic-specific deployment descriptor for the deployed resource adapter.
- *`connection-factory-name`* (required)—defines the logical name that will be associated with this specific deployment of the resource adapter and its corresponding connection factory. The value of this element can be used in other deployed resource adapters through the *`ra-link-ref`* element, allowing multiple deployed Connection Factories to utilize a common deployed resource adapter, as well as share configuration specifications.
- *`description`* (optional)—provides text describing the parent element. This element should include any information that the deployer wants to describe about the deployed Connection Factory.
- *`jndi-name`* (required)—defines the name that will be used to bind the Connection Factory Object into the WebLogic JNDI Namespace. Client EJBs and Servlets use the same JNDI in their defined Reference Descriptor elements of the WebLogic-specific deployment descriptors.
- *`ra-link-ref`* (optional)—allows for the logical association of multiple deployed connection factories with a single deployed resource adapter. The specification of the optional *`ra-link-ref`* element with a value identifying a separately deployed connection factory will result in this newly deployed connection factory sharing the resource adapter that has been deployed with the referenced connection factory. In addition, any values defined in the referred connection factories deployment will be inherited by this newly deployed connection factory unless specified.
- *`native-libdir`* (required if native libraries present)—identifies the directory location to be used for all native libraries present in this resource adapter deployment. As part of deployment processing, all encountered native libraries will be copied to the location specified. It is the responsibility of the administrator to perform the necessary platform actions such that these libraries will be found during WebLogic Server run time.

- *`pool-params`* (optional)—the root element for providing connection pool-specific parameters for this connection factory. WebLogic Server uses these specifications in controlling the behavior of the maintained pool of managed connections.

  Failure to specify this element or any of its specific element items will result in default values being assigned. Refer to the description of each individual element for the designated default value.

- *`initial-capacity`* (optional)—identifies the initial number of managed connections, which WebLogic Server attempts to obtain during deployment.

  Failure to specify this value will result in WebLogic Server using its defined default value.

  Default Value: `1`

- *`max-capacity`* (optional)—identifies the maximum number of managed connections, which WebLogic Server will allow. Requests for newly allocated managed connections beyond this limit results in a *`ResourceAllocationException`* being returned to the caller.

  Failure to specify this value will result in WebLogic Server using its defined default value.

  Default Value: `10`

- *`capacity-increment`* (optional)—identifies the maximum number of additional managed connections that WebLogic Server attempts to obtain during resizing of the maintained connection pool.

  Failure to specify this value will result in WebLogic Server using its defined default value.

  Default Value: `1`

- *`shrinking-enabled`* (optional)—indicates whether or not the connection pool should have unused managed connections reclaimed as a means to control system resources.

  Failure to specify this value will result in WebLogic Server using its defined default value.

  Value Range: `true` | `false`

  Default Value: `true`

- *`shrink-period-minutes`* (optional)—identifies the amount of time the connection pool manager will wait between attempts to reclaim unused managed connections.

  Failure to specify this value will result in WebLogic Server using its defined default value.

  Default Value: `15`

- *`connection-cleanup-frequency`* (optional)—identifies the amount of time the connection pool management will wait between attempts to destroy connection handles which have exceeded their usage duration. This element, used in conjunction with connection-duration-time, prevents connection leaks when an application may have not closed a connection after completing usage.

  Failure to specify this value will result in Weblogic using its defined default value.

  Default Value: `-1`

- *`connection-duration-time`* (optional)—identifies the amount of time a connection can be active. This element, used in conjunction with connection-cleanup-frequency, prevents leaks when an application may have not closed a connection after completing usage.

  Failure to specify this value will result in Weblogic using its defined default value.

  Default Value: `-1`

- *`logging-enabled`* (optional)—indicates whether or not the log writer is set for either the *`ManagedConnectionFactory`* or *`ManagedConnection`*. If this element is set to true, output generated from either the *`ManagedConnectionFactory`* or *`ManagedConnection`* will be sent to the file specified by the *`log-filename`* element.

  Failure to specify this value will result in WebLogic Server using its defined default value.

  Value Range: `true` | `false`

  Default Value: `false`

- *`log-filename`* (optional)—specifies the name of the log file from which output generated from the *`ManagedConnectionFactory`* or a *`ManagedConnection`* is sent.

The full address of the filename is required.

- *map-config-property* (optional, zero or more)—identifies a configuration property name and value that corresponds to an `ra.xml config-entry` element with the corresponding `config-property-name`. At deployment time, all values present in a `map-config-property` specification will be set on the `ManagedConnectionFactory`. Values specified via a map-config-property will supersede any default value that may have been specified in the corresponding `ra.xml` `config-entry` element.
- *map-config-property-name* (optional)—identifies a name that corresponds to an `ra.xml config-entry` element with the corresponding `config-property-name`.
- *map-config-property-value* (optional)—identifies a value that corresponds to an `ra.xml config-entry` element with the corresponding `config-property-name`.
- *security-principal-map* (optional)—provides a mechanism to define appropriate `resource-principal` values for resource adapter and EIS authorization processing, based upon the known WebLogic run time *initiating-principal*. This map allows for the specification of a defined set of initiating principals and the corresponding resource principal's username and password that should be used when allocating managed connections and connection handles.

  A default `resource-principal` can be defined for the connection factory via the map. By specifying an `initiating-principal` value of '*' and a corresponding `resource-principal`, the defined `resource-principal` will be utilized whenever the current identity is *not* matched elsewhere in the map.

  This is an optional element, however, it must be specified in some form if container managed sign-on is supported by the resource adapter and used by *any* client.

  In addition, the deployment-time population of the connection pool with managed connections will be attempted using the defined 'default' resource principal if one is specified.
- *map-entry*—identifies an entry in the `security-principal-map`.
- *initiating-principal* (optional, zero or more)
- *resource-principal* (optional)—can be defined for the connection factory via the *security-principal-map*. By specifying an *initiating-principal*

value of ‘*’ and a corresponding resource-principal, the defined `resource-principal` will be utilized whenever the current identity is *not* matched elsewhere in the map.

- *`resource-username`* (optional)—username identified with the `resource-principal`. Used when allocating managed connections and connection handles.
- *`resource-password`* (optional)—password identified with the `resource-principal`. Used when allocating managed connections and connection handles.

CHAPTER

B Workarounds for Common BEA J2EE Connector Architecture Exceptions

This document provides solutions for two common WebLogic J2EE connector exceptions and provides workarounds for these exceptions:

- "Problem Granting Connection Request to a ManagedConnectionFactory That Does Not Exist in Connection Pool" on page -2
- "ClassCastException" on page -7
- "ResourceAllocationException" on page -8

Problem Granting Connection Request to a ManagedConnectionFactory That Does Not Exist in Connection Pool

The ConnectionPoolManager's `getConnection(ManagedConnectionFactory mcf, ConnectionRequestInfo cxInfo)` method throws this exception internal to WebLogic Server when it is unable to find a ConnectionPool associated with a given ManagedConnectionFactory.

What Causes This Exception? How Can it Be Resolved?

Two causes exist for this exception, as well as one related behavior, which can be confusing:

- Cause Number One: Client-modified ManagedConnectionFactory is Not Hashed on the Server Such That It Can Be Found Again on Subsequent Lookups
- Cause Number Two: A Client is Attempting to Use a Resource Adapter from a Remote JVM
- Related Behavior: Client-side Mutators Do Not Work as Expected

Cause Number One: Client-modified ManagedConnectionFactory is Not Hashed on the Server Such That It Can Be Found Again on Subsequent Lookups

This issue is caused by a combination of the basic behavior of Hashtables in Java and how Serializable objects work with JNDI.

Hashtables allow arbitrary key-value pairs to be stored in memory and found quickly later using the key from the key-value pair. When a key and its associated object is written into a Java Hashtable, the key's `hashCode()` method is invoked to obtain an integer value that is not guaranteed to be unique but is guaranteed to be well distributed with respect to all of the Hashtable keys.

This hashing occurs once when an object is written into the Hashtable. The server writes this object into the data structure with the purpose of later using this same derived integer value repeatedly as a method for finding a small candidate set of matching keys quickly.

When the server performs a subsequent lookup into the Hashtable, it hashes the requested key using its overridden `hashCode()` method or the one found in `java.lang.Object.` Note that the default `hashCode()` method simply returns the memory address of the object whose `hashCode()` method was invoked. It then obtains the set of keys in the backing data structure whose hashed values match the lookup key's hashed key value. Then, the implementation iterates through this candidate list and determines whether there is an appropriate match by executing the following code:

```
for (Entry e = tab[index] ; e != null ; e = e.next) {

    if ((e.hash == hash) && e.key.equals(key)) {

        return e.value;

    }

}

return null;
```

This code compares the lookup key to each candidate list by calling the `equals()` method on each candidate key. If it finds a match, it returns the value. If it falls through the loop without finding a match, it returns a null value.

Consider the object interactions in the deployment of a resource adapter and a JNDI lookup of a resource adapter's ConnectionFactory implementation when the application server only supports Serializable JNDI entries (as opposed to Referencable entries). On deployment of a resource adapter, the application server creates an instance of the ManagedConnectionFactory associated with that resource adapter. The server further associates its ConnectionManager implementation with the ManagedConnectionFactory it just created.

Next, the ManagedConnectionFactory creates an instance of the ConnectionFactory and the ConnectionFactory associates itself with its ManagedConnectionFactory (MCF). At this point, the critical triad of resources is set up. Internally, this link is represented as a pair of Hashtables—one that maps JNDI names to MCFs (`jndiToMCFMap`) and another that maps MCFs to internal representations of connection pools (`poolTable`). The ConnectionManager is implicit to the application server so no explicit mapping is required.

The ConnectionFactory (what a resource adapter's client sees) has an MCF, and the MCF has a ConnectionManager implementation courtesy of the application server (through the implicit mapping mentioned above). Finally, the ConnectionFactory (the client-facing object) is bound into JNDI. This is where the confusion may occur.

The current WebLogic Server JNDI implementation supports `java.io.Serializable` objects but not `javax.naming.Referenceable` ones. When Referenceable objects are bound to JNDI, a reference (containing an endpoint) to the actual object is actually bound to JNDI. The binding does not actually transport the object into the naming tree. When the server binds Serializable objects into the JNDI tree, the object (and all of its referenced Serializable objects) is literally serialized into the tree.

When a client performs a `lookup()` on its naming context for the ConnectionFactory, the same serialization process happens in reverse. The ConnectionFactory implementation is serialized into the address space of the client. Even if the client is running in the same address space as the JNDI implementation (the server), this serialization takes place. The application component (client) now has a *copy* of the actual object graph in the JNDI tree.

Note the MCF is serialized to the client as part of the object graph. Suppose a resource adapter's MCF contains fields used to manage the state of the MCF. Further, assume that the MCF's overridden `hashCode()` and `equals()` methods take these fields into account. For an example, consider a debug flag.

Note that when the resource adapter is deployed, an entry is made into the `jndiToMCFMap` Hashtable. Recalling how Hashtables work in Java, the hash is calculated using all fields, including our debug flag since those fields are used in the `hashCode()` method. When a client performs a JNDI `lookup()`, it obtains its own copy of the CF (and its MCF).

Next, the client sets the debug flag to `true` and calls `getConnection()` on the CF. The CF then calls `allocateConnection()` on the CM with the MCF and the ConnectionRequestInfo objects as parameters. The CM attempts to look up the MCF in its `poolTable` Hashtable using the MCF as the key for the lookup. The Java

Hashtable implementation hashes the MCF (including the debug flag) to obtain a candidate list of matches in the `poolTable` Hashtable. Since the internal state of the MCF has changed between the original deploy-time `put` into the Hashtable and this subsequent lookup, the MCF is not found and the following exception is logged:

```
Problem granting connection request to a
ManagedConnectionFactory which does not exist in connection pool.
Check your MCF's hashcode().
```

Preventing the Manifestation of This Exception

The simplest way to prevent this exception is to have WebLogic Server support JNDI objects that implement the `javax.naming.Referenceable` interface. This requires no additional work on the part of the resource adapter developer. This functionality is planned for the next major release of the product.

In the meantime, developers can take a number of steps to avoid this exception, depending on the resource adapter's MCF state requirements. The easiest solution is to implement the `hashCode()` and `equals()` methods for the resource adapter's MCF. Of course, if you—based on the information in this document—choose to solve the problem in another resource adapter-specific way, that is fine too.

The guideline for implementing `hashCode()` and `equals()` is that these methods should only consider the internal state that is absolutely necessary to uniquely identify the resource adapter instance from any other resource adapter instances running in the server. For example, if a resource adapter is built to talk to a mainframe TP instance, then it might consider its host IP address, port, and region in its `hashCode()` and `equals()` methods. It is also desirable to have something else in the method that can break a tie between two different types of resource adapters that happen to use the same configuration data. This could be a name, class type, and so on.

Cause Number Two: A Client is Attempting to Use a Resource Adapter from a Remote JVM

Cause number two for this exception is a simpler case. The J2EE Connector Architecture is a model that is—at least for the time being—not intended to be accessed remotely. None of the defined interfaces are remote, and the architected system contracts presume a local relationship between an MCF and a ConnectionManager.

Having said that, the fact that WebLogic Server's J2EE Connector Architecture implementation reports an attempt to access a ConnectionFactory remotely as being a problem with an MFC's `hashCode()` implementation is a bug and will be fixed in an upcoming release. There will be a better error message.

Related Behavior: Client-side Mutators Do Not Work as Expected

Another side effect of the lack of a Referenceable JNDI implementation is that since there are two copies of the MCF during a client interaction, client-side changes made to an MCF are not reflected on the server side.

Note that during deployment of a resource adapter, WebLogic Server binds an object graph into the JNDI tree. Also, when a client looks up this graph with `Naming.lookup()`, it obtains a copy of this graph that is serialized—not the original. If a client changes the internal state of an MCF, those changes are not reflected on the server side. Using the debug flag again, if a client executes a `setDebug(true)` method, that change to the state (debug) of the MCF is local to that client's copy of the MCF and the server-side copy will not share the same state as the client.

ClassCastException

Currently, the WebLogic J2EE Connector Architecture container uses a custom classloader to allow for some extended features related to loading and finding classes used by resource adapters. One problem with the current implementation is that referencing resource adapter classes from WebLogic Server containers such as the Web (servlet/JSP) or EJB containers does not behave intuitively. When performing type comparison and casting, the JVM prepends the classloader that loaded the class to the class type (in other words, `com.acb.ra.MyCF` is represented internally as `RARClassloader@abcdef:com.acb.ra.MyCF`). For detailed information, see the Java Language Specification.

The Web and EJB containers use a classloader that is inherited from a WebLogic Server classloader, so it works well with other WebLogic Server resources and is capable of hot deploy and other features. The RARClassLoader—used by the J2EE Connector Architecture—inherits from the Java classloader and is not in the WebLogic hierarchy. This will be fixed at the earliest possible opportunity.

When an application component (running in the Web or EJB container) looks up a ConnectionFactory in JNDI, the returned object is an instance of the object created by the RARClassloader (`RARClassloader@abcdef.com.acb.ra.MyCF`). The object the application component expects is actually `WebLogicClassloader@fedcba:com.acb.ra.MyCF`. When you try to assign a variable or cast, a `ClassCastException` is thrown.

Preventing the Manifestation of This Exception

The most reliable way to avoid this error is to place the resource adapter's classes in the WebLogic Server classpath by adding the path to the resource adapter's classes into the CLASSPATH setting in the startup script for WebLogic Server. This disables the hot deploy and redeploy features that WebLogic Server provides. Again, we recognize that this is not desirable and intend to fix it as soon as possible.

ResourceAllocationException

BlackBoxNoTx > ResourceAllocationException of javax.resource.spi.EISSystemException: SQLException: No suitable driver on createManagedConnection.

The BlackBox resource adapter is a resource adapter that provides a JDBC connection to any database. To avoid this exception, follow these steps to configure this resource adapter to point to a particular database:

1. Make sure that a JDBC connection pool is set up in WebLogic Server for your database.
2. In that RAR, set the `CONNECTION_URL` configuration property, in the `weblogic-ra.xml` file, to the appropriate JDBC pool that you configured in step 1 above.
3. Make sure that the database's JAR file is specified on the WebLogic Server classpath.

BEA ships a BlackBox sample that is configured to use Cloudscape as part of the WebLogic 6.1 kit. Refer to the BlackBox sample for an example on how to set up this configuration.