BEA WebLogic Server™

Programming WebLogic Server J2EE Connectors

Release 8.1
Document Revised: October 29, 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Programming WebLogic Server J2EE Connectors

Part Number	Document Revised	Software Version
N/A	October 29, 2002	BEA WebLogic Server Version 8.1

Contents

3. Transaction Management

4. Connection Management

5. Configuration

A. Troubleshooting

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 WebLogic J2EE Connectors," 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, "Packaging and Deploying Resource Adapters," provides an overview of resource adapters and explains how to package 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 A, "Troubleshooting," provides a solution for a common exception.

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 Final Release—http://java.sun.com/j2ee/download.html#connectorspec
- J2EE Platform Specification, Version 1.3 Final Release—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. *Example*s: 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 WebLogic J2EE Connectors

The following sections provide an overview of BEA WebLogic J2EE Connectors:

J2EE Connector Architecture Terminology

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

- Application Component—can be a server-side component, such as an EJB, JSP, or servlet, that is deployed, managed, and executed on an application server. It can also be a component executed on the Web-client tier but made available to the Web-client by an application server. Examples of the latter type of application component include a Java applet and a DHTML page.
- Caller Principal—a principal that is associated with an application component instance during a method invocation. For example, an EJB instance can call the `getCallerPrincipal` method to get the principal associated with the current security context.

- 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.
- Connection—provides connectivity to a resource manager and enables an application client to connect to a resource manager, perform transactions, and access services provided by that resource manager. A connection can be either transactional or non-transactional. Examples include a database connection and an SAP R/3 connection.
- 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.
- Credential—contains or references security information that can authenticate a principal to additional services. A principal acquires a credential upon authentication or from another principal that allows its credential to be used: the latter is termed principal delegation.
- 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
- Enterprise Information System (EIS) resource—provides EIS-specific functionality to its clients. Examples of an EIS resource include:

  - 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

- Initiating Principal—the security principal representing the end-user that interacts directly with the application. An end-user can authenticate using either a Web client or an application client.
- 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.
- Principal—an entity that can be authenticated by an authentication mechanism deployed in an enterprise. A principal is identified using a principal name and authenticated using authentication data. The content and format of the principal name and the authentication data depend upon the authentication mechanism.
- RAR—resource adapter archive. A compressed (`.zip`) file used to load classes and other files required to run a resource adapter.

- `ra.xml`—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.
- Resource Principal—a security principal under whose security context a connection to an EIS instance is established.
- Security Attributes—a principal has a set of security attributes associated with it. These are related to the authentication and authorization mechanisms. Examples are security permissions and credentials for a principal.
- 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`—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.

Note: BEA WebLogic Server 7.0 is completely compliant with J2EE 1.3. Also, since J2EE is backward compatible, you can still run J2EE 1.2 on WebLogic Server 7.0.

J2EE Connector Architecture Components

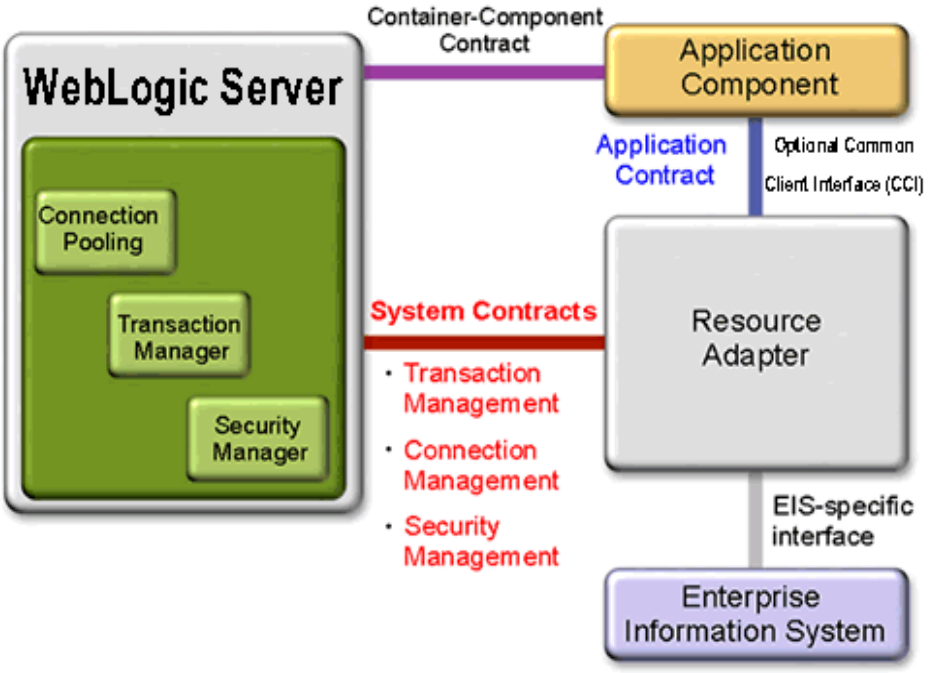
The J2EE Connector Architecture is implemented in an application server such as WebLogic Server and an EIS-specific resource adapter. A resource adapter is a system library specific to an EIS and provides connectivity to the EIS. A resource adapter is analogous to a JDBC driver. The interface between 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 Chapter 4, "Connection Management."

- 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.

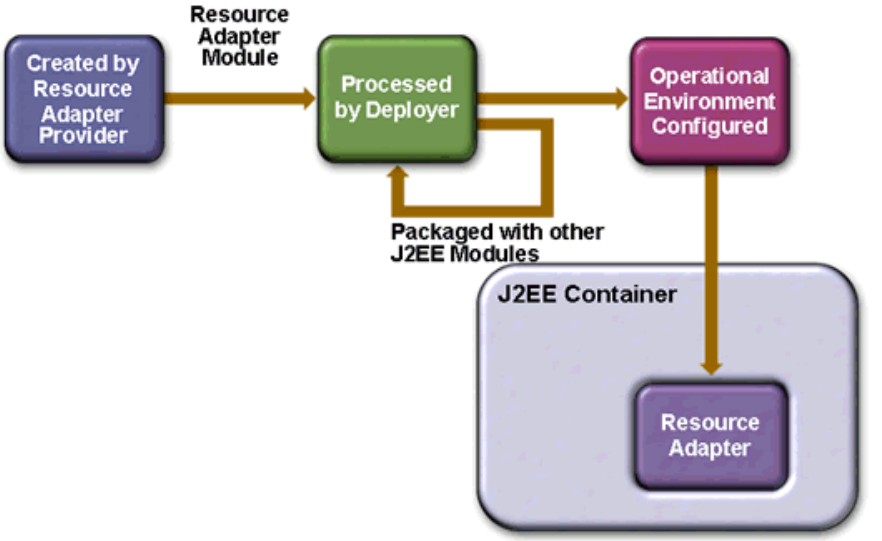
The 1.0 version of the J2EE Connector Architecture recommends that a resource adapter support CCI as its client API, while it requires that the resource adapter implement the system contracts. A resource adapter may choose to have a client API different from CCI, such as the client API based on the Java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



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 7, "Packaging and Deploying Resource Adapters."

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 PointBase 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

As specified in the J2EE Connector Specification, Version 1.0 Final Release, 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.

Password Credential Mapping Mechanism

The J2EE Connector specification, Version 1.0 Final Release requires storage of credentials in a `javax.security.auth.Subject`; the credentials are passed to either the `createManagedConnection()` or `matchManagedConnection()` methods of the `ManagedConnectionFactory` object.

To comply with this, the WebLogic Server J2EE Connector Architecture builds the Subject and stores the credentials by performing the following steps:

1. Instantiate a `weblogic.security.Service.EISResource` object as follows:

   EISResource(java.lang.String applicationName, java.lang.String moduleName, java.lang.String eisName)

2. Obtain the Initiating Principal for the connection request.
3. Obtain the Credentials for that Initiating Principal as follows:

   weblogic.security.Service.PrincipalAuthenticator(String initiatingPrincipal, weblogic.security.Service.Resource eisResource)

4. Instantiate a `javax.security.auth.Subject`.
5. Add the Credentials to the private set in the Subject as follows:

   Subject.getPrivateCredentials().add(Credential)

Authentication Mechanisms

WebLogic Server users must be authenticated whenever they request access to a protected WebLogic Server resource. For this reason, each user is required to provide a credential (a username/password pair or a digital certificate) to WebLogic Server. The following types of authentication mechanisms are supported by WebLogic Server:

- Password authentication—a user ID and password are requested from the user and sent to WebLogic Server in clear text. WebLogic Server checks the information and if it is trustworthy, grants access to the protected resource.

  The SSL (or HTTPS) protocol can be used to provide an additional level of security to password authentication. Because the SSL protocol encrypts the data transferred between the client and WebLogic Server, the user ID and password of the user do not flow in the clear. Therefore, WebLogic Server can authenticate the user without compromising the confidentiality of the user's ID and password.

- JAAS authentication—JAAS is a standard extension to the security in the Java Software Development Kit version 1.3. JAAS provides the ability to enforce access controls based on user identity. JAAS is provided in WebLogic Server as an alternative to the JNDI authentication mechanism.

  WebLogic Server uses the authentication portion of the standard JAAS. The JAAS LoginContext provides support for the ordered execution of all configured authentication provider LoginModule instances and is responsible for the management of the completion status of each configured provider.

  Note: JAAS is the preferred method of authentication, however, the WebLogic-supplied LoginModule only supports username and password authentication. Thus, for client certificate authentication (also referred to as two-way SSL authentication), you should use JNDI. To use JAAS for client authentication, you must write a custom LoginModule that does certificate authentication.

For more information, see the following sections in *Managing WebLogic Security*:

- "Configuring the SSL Protocol"
- "Writing a Client Application using JAAS Authentication"

Defining Users and Groups

The following sections discuss the definition of users and groups. For more information on how to create users and groups, see *Managing WebLogic Security*.

Users

Users are entities that can be authenticated in a WebLogic Server security realm. A user can be a person or a software entity, such as a Java client. Each user is given a unique identity within a WebLogic Server security realm. As a system administrator you must guarantee that no two users in the same security realm are identical.

Defining users in a security realm involves specifying a unique name and password for each user that will access resources in the WebLogic Server security realm in the users window of the Administration Console.

Three special users are provided for use by resource adapters. They are as follows:

- `wls_ra_initial`—If you define a mapping for this user, the specified credentials are used for the initial connections created when starting the connection pool for this resource adapter. The `InitialCapacity` parameter on the pool specifies the number of initial connections. If you do not define a mapping for this user the default mapping `wls_ra_default` (if provided) is used. Otherwise, no credentials are provided for the initial connections.
- `wls_ra_anonymous`—If you define a mapping for this user, the specified credentials are used when no user is authenticated for the connection request on the resource adapter.
- `wls_ra_default`—If you define a mapping for this user, the specified credentials are used when no other mapping applies for the current user or when no anonymous mapping is provided in the case where there is no authenticated user.

Groups

A group represents a set of users who usually have something in common, such as working in the same department in a company. Groups are a means of managing a number of users in an efficient manner. You grant users and groups security roles. These security roles are used to create a security policy, which restricts access to server resources. For more information, see *"Setting Protections for WebLogic Resources."*

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.

Security Policy Processing

The J2EE Connector Specification, Version 1.0 Final Release defines default security policies for any resource adapters running in an application server. It also defines a way for a resource adapter to provide its own specific security policies overriding the default.

In compliance with this specification, WebLogic Server dynamically modifies the runtime environment for resource adapters. If the resource adapter has not defined specific security policies, WebLogic Server overrides the runtime environment for the resource adapter with the default security policies specified in the J2EE Connector Architecture Specification. If the resource adapter has defined specific security policies, WebLogic Server first overrides the runtime environment for the resource adapter first with a combination of the default security policies for resource adapters and the specific policies defined for the resource adapter. Resource adapters define specific security policies using the `security-permission-spec` element in the `ra.xml` deployment descriptor file.

For more information on security policy processing requirements, see the “Security Permissions” section of the “Runtime Environment” chapter in the J2EE Connector Specification, Version 1.0 Final Release (http://java.sun.com/j2ee/download.html#connectorspec).

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

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 Final Release):

- 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 Final Release (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.

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 Final Release, 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 Final Release, 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-frequency-seconds` element in the `weblogic-ra.xml` descriptor file to identify the amount of time (in seconds) the Connection Pool Management will wait between attempts to reclaim unused Managed Connections.. The default value of this element is `900` minutes.

Detecting Connection Leaks

Connection leaks result from faulty application components, such as an Enterprise Javabean (EJB), not doing their job to close a connection after using them. As stated in the J2EE Connector Specification, Version 1.0 Final Release, 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 be exhausted of its available connections, and future connection requests can therefore fail.

WebLogic Server provides two mechanisms for preventing this scenario:

- Leveraging a garbage collector
- Providing an idle timer for tracking the usage of connection objects

Garbage Collector Method

WebLogic Server automatically detects connection leaks by leveraging its Java Virtual Machine (JVM) garbage collector mechanism. When an application component terminates and the connections it uses become dereferenced, the garbage collector calls the connection object's `finalize()` method.

When the garbage collector calls the `finalize()` method, if WebLogic Server determines the application component has not closed the connection, the server automatically closes the connection by calling the resource adapter's `ManagedConnection.cleanup()` method; WebLogic Server behaves as it would had it received a `CONNECTION_CLOSED` event upon proper closure of the application component connection.

Idle Timer Method

Because the garbage collector does not behave in a predictable manner and may in fact never be called, WebLogic Server provides a second connection leak detection method, the idle timer. The idle timer allows WebLogic Server to track the last time each connection was used. To configure the idle timer for each connection to an EIS, refer to Appendix A, "weblogic-ra.xml Deployment Descriptor Elements."

When an application component obtains a connection for usage but is not actively

using it, the idle timer starts ticking. As a precaution against closing a connection that is actually active, when a connection has reached its configured maximum limit, WebLogic Server does not automatically close the connection. Instead, WebLogic Server waits to close the connection that has exceeded its idle time until it is absolutely necessary to do so.

If the connection pool for a resource adapter has exceeded its maximum number of allocated connections and there are no allocated connections in the free pool, a connection request fails. At times, connections exist that have been leaked and not been put back on the free pool, even though they are inactive. In this scenario, WebLogic Server closes connections that have exceeded their maximum idle time at the time of a connection request so that the request succeeds.

Monitoring Connections Using the Console

The BEA J2EE Connector Architecture provides you with monitoring capabilities in the WebLogic Server Console that show detected leaks and provides a method for looking up stacks to determine which application(s) is causing the leak. Delete buttons in the Console allow you to dynamically close leaked connections that are identified; the option to delete connections is only available for connections that have exceeded their specified idle time and are safe to delete (in other words, the connection is not involved in a transaction).

The `connection-profiling-enabled` element of the `weblogic-ra.xml` file indicates whether or not the connection pool should store the call stacks of where each connection is allocated. If you set this element value to `true`, you can view this information on active connections through the Console. Also, you can view the stacks for leaked and idle connections, and you can debug components that fail to close connections.

Getting Started

There are two methods for accessing monitoring tools using the Console.

Method One

1. In the left pane of the Console, select Deployments > Connectors to display a list of connectors.
2. Right-click a connector, 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.

Method Two

1. In the right pane of the Console, under Deployments, select Connectors.

   A connector table is displayed.
2. Under the Name column, click the connector to monitor.
3. In the Monitoring tab, select Monitor all Connector Connection Pool Runtimes.

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

Viewing Leaked Connections

A Connection Leak Profiles column in the Console allows you to view profile information pertaining to leaked connections. This column is not to be confused with the Leaked Connections Detected column, which simply displays the number of leaked connections.

A key difference between these two columns is the Connection Leak Profiles column is controlled by use of the `connection-profiling-enabled` setting in the `weblogic-ra.xml` file. By default, this setting is `false`, so normally the Connection Leak Profiles column will be zero (disabled). However, the Leaked Connections Detected column is always enabled and will always display the number of leaked connections.

There are two methods for viewing leaked connections using the Console.

Method One

1. In the left pane of the Console, select Deployments > Connectors to display a list of connectors.
2. Right-click a connector, and select View Leaked Connections from the pop-up menu.

   Connection pool run-time information for the selected connector is provided in the right pane.
3. Under the Connection Leak Profiles column, click the number of leaked connections pertaining to the selected connector.

   Leaked connection information is displayed in the right pane.

Method Two

1. In the right pane of the Console, under Deployments, select Connectors.

   A connector table is displayed.
2. Under the Name column, click the name of the connector to monitor.
3. In the Monitoring tab, select Monitor all Connector Connection Pool Runtimes.

   Connection pool run-time information for the selected connector is provided in the right pane.
4. Under the Connection Leak Profiles column, click the number of leaked connections pertaining to the selected connector.

   Leaked connection information is displayed in the right pane.

Viewing Idle Connections

A Connection Idle Profiles column in the Console allows you to view profile information pertaining to idle connections. This column is not to be confused with the Idle Connections Detected column, which simply displays the number of idle connections.

A key difference between these two columns is the Connection Idle Profiles column is controlled by use of the `connection-profiling-enabled` setting in the `weblogic-ra.xml` file. By default, this setting is `false`, so normally the Connection Idle Profiles column will be zero (disabled). However, the Idle Connections Detected column is always enabled and will always display the number of idle connections.

There are two methods for idle connections using the Console.

Method One

1. In the left pane of the Console, select Deployments > Connectors to display a list of connectors.
2. Right-click a connector, and select View Idle Connections from the pop-up menu.

   Connection pool run-time information for the selected connector is provided in the right pane.
3. Under the Connection Idle Profiles column, click the number of idle connections pertaining to the selected connector.

   Idle connection information is displayed in the right pane.

Method Two

1. In the right pane of the Console, under Deployments, select Connectors.

   A connector table is displayed.
2. Under the Name column, click the name of the connector to monitor.
3. In the Monitoring tab, select Monitor all Connector Connection Pool Runtimes.

   Connection pool run-time information for the selected connector is provided in the right pane.
4. Under the Connection Idle Profiles column, click the number of idle connections pertaining to the selected connector.

   Idle connection information is displayed in the right pane.

Deleting Connections

To delete leaked or idle connections using the Console:

1. In the right pane of the Console, under Deployments, select Connectors.

   A connector table is displayed.

2. Under the Name column, click the name of the connector to monitor.

3. In the Monitoring tab, select Monitor all Connector Connection Pool Runtimes.

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

4. Under the Connections column, click the number of connections pertaining to the selected connector.

   Connection information is displayed in a table format, each row representing a single connection.

5. Click the Delete button to the right of a connection to delete it.

Error Logging and Tracing Facility

As stated in the J2EE Connector Specification, Version 1.0 Final Release, 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.

The `weblogic-ra.xml` file descriptor file supports two elements that allow configuration of logging and tracing for resource adapters deployed in WebLogic Server. These elements are as follows:

- The `logging-enabled` element indicates whether logging is enabled or disabled for a specific `ManagedConnectionFactory` at deployment time. The default value for this element is `false`.
- The `log-filename` element specifies the filename in which to write the logging information that the `ManagedConnectionFactory` produces.

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

CHAPTER

5 Configuration

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

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.

WebLogic Builder

WebLogic Builder is a visual environment for editing an application's deployment descriptor XML files. You can view descriptor files while you visually edit them in WebLogic Builder, and you won't need to make textual edits to the XML files. For more information, see the WebLogic Builder Online Help.

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 the BEA Developer Center.

Configuring Resource Adapters

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

Resource Adapter Overview

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 Final Release. 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 Archive (RAR)

The following are the main steps for creating a resource adapter archive (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, Final Release (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 using a standard compiler for the interfaces and implementation into class files.

   For instructions on compiling, refer to "Compiling Java Code" in Chapter 2, "Developing WebLogic Server Applications" of *Developing WebLogic Server J2EE Applications*.

3. Create the resource connector-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 detailed information about creating connector-specific deployment descriptors, refer to Appendix A, "weblogic-ra.xml Deployment Descriptor Elements."

4. Package the Java classes into a Java archive (JAR) file.

The first step in creating a JAR file is to create a connector staging directory anywhere on your hard drive. Place the JAR file in the staging directory and the deployment descriptors in a subdirectory called `META-INF`.

Then you create the resource adapter archive by executing a `jar` command similar to the following in the staging directory:

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

For detailed information about creating the resource adapter RAR archive file, refer to "Packaging" in *Developing WebLogic Server Applications*.

5. Auto-deploy the RAR resource adapter archive file on WebLogic Server for testing purposes.

   For detailed information about auto-deploying components and applications, refer to "Tools for Deploying" in *WebLogic Server Deployment and Packaging*.

   While you are testing the resource adapter, you might need to edit the resource adapter deployment descriptors. You can do this manually or use WebLogic Builder.

   For detailed information, refer to *WebLogic Builder Online Help*. See Appendix A, "weblogic-ra.xml Deployment Descriptor Elements"for detailed information on the elements in these deployment descriptors.

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 *WebLogic Server Deployment and Packaging* 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 anywhere on your hard drive 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.rar
          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.

   Refer to Appendix A, "weblogic-ra.xml Deployment Descriptor Elements" for more information on the `weblogic-ra.xml` DTD.

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.rar -C c:/stagedir
```

7. Deploy the resource adapter to WebLogic Server. For detailed information about deploying components and applications, refer to "Tools for Deploying" in *WebLogic Server Deployment and Packaging*.

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

Configuring the weblogic-ra.xml File

In addition to supporting features of the standard resource adapter configuration `ra.xml` file, BEA WebLogic Server defines an additional deployment descriptor file, the `weblogic-ra.xml` file. This file contains parameters that are specific to configuring and deploying resource adapters in WebLogic Server. 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. 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.

In the `weblogic-ra.xml` 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.
- Values for configuration properties defined in a `<config-entry>` element of the J2EE resource adapter deployment descriptor, `ra.xml`.
- Flag to indicate whether logging is required for the ManagedConnectionFactory or ManagedConnection.
- File to store logging information for the ManagedConnectionFactory or ManagedConnection.
- The amount of time a connection can remain idle.
- Whether to store call stacks of where each connection is allocated.

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."

Automatic Generation of the weblogic-ra.xml File

In WebLogic Server, a resource adapter archive (RAR) must include a `weblogic-ra.xml` deployment descriptor file in addition to the `ra.xml` deployment descriptor file specified in the J2EE Connector 1.0 specification. However, if a resource adapter is deployed in WebLogic Server without a `weblogic-ra.xml` file, a template `weblogic-ra.xml` file populated with default element values is automatically added to the resource adapter archive. This automatic resource file generation simplifies the process of establishing the parameters necessary to deploy the resource adapter in WebLogic Server.

If your 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 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-frequency-seconds>900</shrink-frequency-seconds>
               </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, JARs, 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 Appendix A, "weblogic-ra.xml Deployment Descriptor Elements."

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 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 Final Release (http://java.sun.com/j2ee/download.html#connectorspec).

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 Final Release—http://java.sun.com/j2ee. The following sections correspond to the System Contract requirements identified in this specification:

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 Final Release (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 Final Release (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 Final Release (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.

CHAPTER

7 Packaging and Deploying Resource Adapters

This chapter discusses packaging and deploying requirements for resource adapters and provides instructions for performing these tasks.

Packaging Resource Adapters

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

This section discusses resource adapter packaging guidelines, requirements and limitations, and provides instructions for packaging resource adapters.

Packaging Directory Structure

A resource adapter is a WebLogic Server component contained in a resource adapter archive (RAR) within the `applications/` directory. The deployment process begins with the RAR 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 a RAR 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 a RAR. A resource adapter is structured as in the following example:

Listing 7-1 Resource Adapter Directory Structure

```
/META-INF/ra.xml
/META-INF/weblogic-ra.xml
/META-INF/MANIFEST.MF (optional)
/images/ra.jpg
/readme.html
/eis.jar
/utilities.jar
/windows.dll
/unix.so
```

Packaging Considerations

The following are packaging requirements for resource adapters:

- Deployment descriptors (`ra.xml` and `weblogic-ra.xml`) must be in a subdirectory called `META-INF`.
- An optional `MANIFEST.MF` also resides in `META-INF`. A manifest file is automatically generated by the JAR tool and is always the first entry in the JAR file. By default, it is named `META-INF/MANIFEST.MF`. The manifest file is the place where any meta-information about the archive is stored. For more information, see http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/jar.html.
- The resource adapter can contain multiple JARs 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`)
- Ensure that any dependencies of a resource adapter on platform-specific native libraries are resolved.
- When a standalone resource adapter RAR is deployed, the resource adapter must be made available to all J2EE applications in the application server.
- When a resource adapter RAR packaged within a J2EE application EAR is deployed, the resource adapter must be made available only to the J2EE application with which it is packaged.
- A resource adapter deployed in WebLogic Server supports the `CLASSPATH` entry in `MANIFEST.MF` to reference a class or resource such as a property.

For more information on packaging requirements, refer to chapter 10 of the J2EE Connector Specification, Version 1.0 Final Release (`http://java.sun.com/j2ee/download.html#connectorspec`).

Packaging Limitations

The following are WebLogic Server packaging limitations on resource adapters:

- 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.
- The WebLogic J2EE Connector Architecture does not allow you to reload a standalone resource adapter without reloading the client that is using it. (This limitation is due to the J2EE Connector Specification, Version 1.0 limitation of not providing a remotable interface.)
- The ConnectionPoolManager's `getConnection(ManagedConnectionFactory mcf, ConnectionRequestInfo cxInfo)` method throws an exception internal to WebLogic Server when it is unable to find a ConnectionPool associated with a given ManagedConnectionFactory. For more information, see Appendix A, "Troubleshooting."

Packaging Resource Adapters Archives (RARs)

After you stage one or more resource adapters in a directory, you package them in a Java Archive (JAR). Before you package your resource adapters, be sure you read and understand the chapter entitled "WebLogic Server Application Classloading" in *Developing WebLogic Server Applications*, which describes how WebLogic Server loads classes.

To stage and package a resource adapter:

1. Create a temporary staging directory anywhere on your hard drive.
2. Compile or copy the resource adapter Java classes into the staging directory.
3. Create a JAR to store the resource adapter Java classes. Add this JAR 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 the resource adapter classes and deployment descriptors are set up in the staging directory, you can create the RAR with a JAR command such as:

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

   This command creates a RAR that you can deploy on a WebLogic Server or package in an enterprise application archive (EAR).

   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 are relative to the directory where you staged the resource adapters.

For more information on this topic, see "Creating and Modifying Resource Adapters: Main Steps" on page 5-3.

Deploying Resource Adapters

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:

- Using the WebLogic Server Administration Console.
- Using the weblogic.Deployer Utility.
- Using auto-deployment. This is useful for testing purposes. For more information,

For more information on these tools, see "Deployment Tools Reference" in *Deploying WebLogic Server Applications*.

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 Final Release. 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."

You can modify deployment descriptors using the following tools:

- Using WebLogic Builder. WebLogic Builder is a WebLogic Server tool for generating and editing deployment descriptors for J2EE applications. It can also deploy applications to single servers.For more information, see "Deployment Tools Reference" in *Deploying WebLogic Server Applications*.
- Manually using a simple XML editor. BEA provides a 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 the BEA Developer Center. You can also manually edit a deployment descriptor using a simple text editor.

Resource Adapter Deployment Names

When you deploy a resource adapter archive (RAR) 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 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; you can use the same deployment name to redeploy the resource adapter at a later time.

CHAPTER

8 Client Considerations

The following sections discuss WebLogic J2EE Connector Architecture client considerations:

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 Final Release 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 Final Release 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 Final Release 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 Final Release 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.

CHAPTER

A weblogic-ra.xml Deployment Descriptor Elements

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

If your resource adapter archive (RAR) does not contain a `weblogic-ra.xml` file, WebLogic Server automatically generates this file for you.

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 weblogic-connection-factory-dd PUBLIC "-//BEA Systems, Inc.//DTD WebLogic 8.1.0 Connector//EN" "http://www.bea.com/servers/wls810/dtd/weblogic810-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/wls810/dtd/weblogic810-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 WebLogic Builder to Edit Deployment Descriptors

WebLogic Builder provides a visual environment for editing an application's deployment descriptor XML files. You can view these XML files as you visually edit them in WebLogic Builder, but you won't need to make textual edits to the XML files.

Use WebLogic Builder for the following development tasks:

- Generate deployment descriptor files for a J2EE module
- Edit a module's deployment descriptor files
- View deployment descriptor files
- Compile and validate deployment descriptor files
- Deploy a module to a server

For instructions on using WebLogic Builder, refer to the WebLogic Builder documentation.

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`

(deprecated) shrink-period-minutes (optional)

This is a deprecated element. It has been replaced by the `shrink-frequency-seconds` element.

Identifies the amount of time the connection pool manager will wait between attempts to reclaim unused managed connections.

Default Value: `15`

connection-cleanup-frequency (optional)

This is a deprecated element. 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`

Note: The `connection-cleanup-frequency` element is a deprecated element. If you currently have this parameter in your configuration, you will still be able use deployment functions. However, this element will have no effect on the configuration.

connection-duration-time (optional)

This is a deprecated element. 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`

Note: The `connection-duration-time` element is a deprecated element. If you currently have this parameter in your configuration, you will still be able use deployment functions. However, this element will have no effect on the configuration.

(deprecated) connection-maxidle-time (optional)

This is a deprecated element. It has been replaced by the `inactive-connection-timeout-seconds element`.

Identifies the amount of time (in seconds) a connection handle can remain idle. This element prevents leaks when an application may have not closed a connection after completing usage. Idle connections will only be terminated in the case where the connection pool becomes full, and a new connection request is about to fail because of this.

Default Value: 0

connection-profiling-enabled (optional)

Indicates whether or not the connection pool should store the call stacks of where each connection is allocated. If enabled this information can be viewed on active connections through the Console. Also, the stacks for Leaked and Idle connections will be available if this is enabled and can help debug components that fail to close connections.

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

Value Range: `true` | `false`

Default Value: `false`

shrink-frequency-seconds (optional)

Indentifies the amount of time (in seconds) Connection Pool Management will wait between attempts to reclaim unused Managed Connections.

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

Default Value: `900 seconds`

inactive-connection-timeout-seconds (optional)

Identifies the amount of time (in seconds) a Connection handle can remain inactive. This element prevents leaks when an Application may have not closed a connection after completing usage. Inactive connections will be terminated as soon as they are detected.

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

Default Value: `0`

highest-num-waiters (optional)

The maximum number of waiters that can concurrently block waiting to reserve a connection from the pool.

Default Value: `0`

highest-num-unavailable (optional)

The maximum number of physical connections (Managed Connections) in the pool that can be made unavailable (to the application) for purposes such as refreshing a connection. Note that in cases such as the backend system being unavailable, the specified value might be exceeded due to factors outside of the control of the pool.

Default Value: `0`

connection-creation-retry-frequency-seconds (optional)

The periodicity of retry attempts by the pool to establish connections.

Default Value: `0`

connection-reserve-timeout-seconds (optional)

The number of seconds after which the call to reserve a connection from the pool will timeout.

Default Value: `-1 (do not block when reserving resources)`

test-frequency-settings (optional)

The periodicity at which connections in the pool are tested.

Default Value: `0`

check-on-create-enabled (optional)

Enables testing of newly created connections.

Value Range: `true|false`

Default Value: `false`

check-on-release-enabled (optional)

Enables testing of connections when they are being released back into the pool.

Value Range: `true|false`

Default Value: `false`

check-on-reserved-enabled (optional)

Enables testing of connections when they are being reserved.

Value Range: `true|false`

Default Value: `false`

max-num-match-retries (optional)

Specifies the number of times that the `matchManagedConnections()` method is retried on the resource adapter's Managed Connection object, before WebLogic Server creates a new Managed Connection using its `createManagedConnection()` method.

Default Value: `0`

match-connections-supported (optional)

Indicates whether or not the resource adapter supports the `ManagedConnectionFactory.matchManagedConnections()` method. If the resource adapter does not support this method (always returns null for this method), then WebLogic Server bypasses this method call during a connection request.

Value Range: `true|false`

Default Value: `true`

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)

`This is a deprecated element.` 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

A Troubleshooting

Cannot Map a ManagedConnectionFactory

BEA WebLogic Server writes the following message to the server log file:

Listing B-1 Cannot Map a ManagedConnectionFactory...

```
Cannot map a ManagedConnectionFactory to a Connection pool. Ensure
that the MCF's hashcode() and equals() methods are implemented
properly.
```

Causes and Workarounds

This exception occurs during a `getConnection()` method call from an application component to a resource adapter and can occur due to the following reasons:

- Remote Java Virtual Machine (JVM). The application component is executing in a different JVM than the one that is hosting the resource adapter.
- Improper implementation of ManagedConnectionFactory

Remote JVM

As currently specified, the J2EE Connector Architecture does not provide for remote access. None of the defined interfaces are remote, and the architected system contracts presume a local relationship between a ManagedConnectionFactory and a Connection Manager.

As a result, you must deploy your application so that the application components are hosted in the same Java Virtual Machine as your resource adapters.

Improper Implementation of ManagedConnectionFactory

WebLogic Server depends on the `hashCode()` and `equals()` methods of the resource adapter's ManagedConnectionFactory when WebLogic Server is managing the connections to the resource adapter. The server uses both these methods to identify a unique instance of a ManagedConnectionFactory. As a result, you need to be aware of a few things when implementing these methods in your ManagedConnectionFactory.

For a given instance of a ManagedConnectionFactory, its `hashCode()` method must always return the same value throughout the entire lifetime of that ManagedConnectionFactory. This begins when the associated resource adapter is deployed and ends when it is undeployed.

You must carefully write the `equals()` method of a ManagedConnectionFactory to distinguish between different instances of ManagedConnectionFactory. You are free to use class or instance data that can change during the lifetime of the resource adapter in the `equals()` method. This freedom to use modifiable data in the `equals()` method is new with WebLogic Server 7.0. Prior to the 7.0 release, you were restricted from doing this. BEA has changed the way WebLogic Server stores objects, such as ManagedConnectionFactory objects, in its JNDI tree.

Prior to the 7.0 release, WebLogic Server stored serialized copies of objects in the JNDI tree. For example, when a resource adapter was deployed, an entire copy of its ManagedConnectionFactory object was serialized and stored in the WebLogic Server JNDI tree. The entire state of that object at that time, including the values of all its data members, was copied and stored. Later, when a connection request was made to that resource adapter, WebLogic Server would use the ManagedConnectionFactory passed along in the connection request to try to locate the connection pool to which it had earlier assigned the deployed ManagedConnectionFactory. If the state of the ManagedConnectionFactory object had changed between deploy time and connection request time, and if this state was reflected in the behavior of the `equals()` method,

then the two objects (the one copied into the JNDI tree at deploy time and the one presented with the connection request) were actually different objects, and WebLogic disallowed the connection request.

This is no longer a problem because WebLogic Server stores references to objects in its JNDI tree instead of copies of the objects. Now, when a resource adapter is deployed, only a reference to its ManagedConnectionFactory object is stored into JNDI. Later, when the connection request is made and WebLogic Server uses the stored reference to the deployed ManagedConnectionFactory object, it finds the same object that is now being passed along in the connection request. An invocation of the object's `equals()` method operates on the current state of the object, and it does not matter if the state of the object has changed since deploy time.

Index

Symbols

A

B

C

D

E

I

J

L

M

N

O

P

R

S

T

W

X