BEA WebLogic Server®

Programming WebLogic Resource Adapters

Version 10.0
Revised: March 30, 2007

1. Introduction and Roadmap

Document Scope and Audience. 1-1
Guide to This Document 1-2
Related Documentation 1-3
Examples for the Resource Adapter Developer. 1-3
New and Changed Features in This Release 1-4

2. Understanding Resource Adapters

Overview of Resource Adapters 2-1
Comparing WebLogic Server and WebLogic Integration Resource Adapters. 2-2
Inbound, Outbound, and Bidirectional Resource Adapters 2-2
Comparing 1.0 and 1.5 Resource Adapters 2-3
J2EE Connector Architecture 2-4
J2EE Architecture Diagram and Components 2-4
System-Level Contracts 2-6
Resource Adapter Deployment Descriptors 2-7

3. Creating and Configuring Resource Adapters

Creating and Configuring Resource Adapters: Main Steps. 3-1
Modifying an Existing Resource Adapter 3-3
Configuring the ra.xml File 3-4
Configuring the weblogic-ra.xml File 3-4
Editing Resource Adapter Deployment Descriptors 3-6
Editing Considerations 3-6
Schema Header Information 3-6
Conforming Deployment Descriptor Files to Schema 3-7
Dynamic Descriptor Updates: Console Configuration Tabs 3-8
Dynamic Pool Parameters 3-8

Dynamic Logging Parameters 3-8
Automatic Generation of the weblogic-ra.xml File 3-9
(Deprecated) Configuring the Link-Ref Mechanism 3-9

4. Programming Tasks

Required Classes for Resource Adapters 4-1
Programming a Resource Adapter to Perform as a Startup Class 4-2
Suspending and Resuming Resource Adapter Activity 4-7
Extended BootstrapContext 4-13
Diagnostic Context ID 4-13
Dye Bits 4-14
Callback Capabilities 4-14

5. Connection Management

Connection Management Contract 5-1
Connection Factory and Connection 5-2
Resource Adapters Bound in JNDI Tree 5-2
Obtaining the ConnectionFactory (Client-JNDI Interaction) 5-2
Configuring Outbound Connections 5-4
Connection Pool Configuration Levels 5-4
Multiple Outbound Connections Example 5-5
Configuring Inbound Connections 5-9
Configuring Connection Pool Parameters 5-11
initial-capacity: Setting the Initial Number of ManagedConnections 5-11
max-capacity: Setting the Maximum Number of ManagedConnections 5-12
capacity-increment: Controlling the Number of ManagedConnections 5-12
shrinking-enabled: Controlling System Resource Usage 5-12
shrink-frequency-seconds: Setting the Wait Time Between Attempts to Reclaim Unused ManagedConnections 5-13

highest-num-waiters: Controlling the Number of Clients Waiting for a Connection 5-13
highest-num-unavailable: Controlling the Number of Unavailable Connections . . 5-13
connection-creation-retry-frequency-seconds: Recreating Connections 5-13
match-connections-supported: Matching Connections 5-13
test-frequency-seconds: Testing the Viability of Connections 5-14
test-connections-on-create: Testing Connections upon Creation 5-14
test-connections-on-release: Testing Connections upon Release to Connection Pool 5-14
test-connections-on-reserve: Testing Connections upon Reservation 5-15
Connection Proxy Wrapper - 1.0 Resource Adapters 5-15
Possible ClassCastException 5-15
Turning Proxy Generation On and Off. 5-16
Testing Connections 5-16
Configuring Connection Testing 5-16
Testing Connections in the Administration Console. 5-17

6. Transaction Management

Supported Transaction Levels 6-1
XA Transaction Support. 6-2
Local Transaction Support 6-2
No Transaction Support 6-2
Configuring Transaction Levels 6-3

7. Message and Transactional Inflow

Overview of Message and Transactional Inflow 7-1
Architecture Components 7-2
Inbound Communication Scenario. 7-4
How Message Inflow Works 7-4
Handling Inbound Messages 7-5

Proprietary Communications Channel and Protocol 7-6
Message Inflow to Message Endpoints (Message-driven Beans). 7-6
Deployment-Time Binding Between an MDB and a Resource Adapter 7-6
Binding an MDB and a Resource Adapter 7-7
Dispatching a Message. 7-7
Activation Specifications 7-8
Administered Objects. 7-8
Transactional Inflow 7-9
Using the Transactional Inflow Model for Locally Managed Transactions 7-10

8. Security

Container-Managed and Application-Managed Sign-on 8-1
Application-Managed Sign-on. 8-2
Container-Managed Sign-on 8-2
Password Credential Mapping 8-2
Authentication Mechanisms. 8-3
Credential Mappings 8-3
Creating Credential Mappings Using the Console. 8-5
Security Policy Processing 8-5
Configuring Security Identities for Resource Adapters 8-6
default-principal-name: Default Identity 8-8
manage-as-principal-name: Identity for Running Management Tasks 8-9
run-as-principal-name: Identity Used for Connection Calls from the Connector Container into the Resource Adapter 8-10
run-work-as-principal-name: Identity Used for Performing Resource Adapter Management Tasks 8-10
Configuring Connection Factory-Specific Authentication and Re-authentication Mechanisms 8-11

9. Packaging and Deploying Resource Adapters

Packaging Resource Adapters 9-1

Packaging Directory Structure 9-1

Packaging Considerations 9-2

Packaging Limitation 9-3

Packaging Resource Adapter Archives (RARs). 9-3

Deploying Resource Adapters 9-4

Deployment Options. 9-4

Resource Adapter Deployment Names 9-5

Production Redeployment 9-5

Suspendable Interface and Production Redeployment 9-6

Production Redeployment Requirements 9-6

Production Redeployment Process 9-6

A. weblogic-ra.xml Schema

weblogic-connector A-2

work-manager A-6

security. A-10

default-principal-name A-12

manage-as-principal-name A-12

run-as-principal-name. A-13

run-work-as-principal-name A-13

properties A-13

admin-objects. A-14

admin-object-group A-15

admin-object-instance A-17

outbound-resource-adapter A-18

default-connection-properties. A-19

pool-params A-22
logging A-25
connection-definition-group. A-28
connection-instance A-30

B. Resource Adapter Best Practices

Classloading Optimizations for Resource Adapters B-1
Connection Optimizations. B-2
Thread Management B-2
InteractionSpec Interface. B-2

CHAPTER 1

Introduction and Roadmap

The following sections describe the contents and organization of this guide—*Programming WebLogic Resource Adapters*.

- "Document Scope and Audience" on page 1-1
- "Guide to This Document" on page 1-2
- "Related Documentation" on page 1-3
- "Examples for the Resource Adapter Developer" on page 1-3
- "New and Changed Features in This Release" on page 1-4

Document Scope and Audience

This document is written for resource adapter users, deployers, and software developers who develop applications that include J2EE resource adapters to be deployed to WebLogic Server. This document also contains information that is useful for business analysts and system architects who are evaluating WebLogic Server or considering the use of WebLogic Server resource adapters for a particular application.

The topics in this document are relevant during the design and development phases of a software project. The document also includes topics that are useful in solving application problems that are discovered during test and pre-production phases of a project.

This document does not address production phase administration, monitoring, or performance tuning topics. For links to WebLogic Server documentation and resources for these topics, see "Related Documentation" on page 1-3.

It is assumed that the reader is familiar with J2EE and resource adapter concepts. The foundation document for resource adapter development is Sun Microsystems J2EE Connector Architecture Specification, Version 1.5 Final Release (referred to in this guide as the J2CA 1.5 Specification): http://java.sun.com/j2ee/connector/ . Resource adapter developers should become familiar with the J2CA 1.5 Specification. This document emphasizes the value-added features provided by WebLogic Server resource adapters and key information about how to use WebLogic Server features and facilities to get a resource adapter up and running.

Guide to This Document

- This section, Chapter 1, "Introduction and Roadmap," introduces the organization of this guide.
- Chapter 2, "Understanding Resource Adapters," introduces you to the BEA WebLogic Server implementation of the J2EE Connector Architecture as well as the resource adapter types and XML schema.
- Chapter 3, "Creating and Configuring Resource Adapters," describes how to create resource adapters using the BEA WebLogic Server implementation of the J2EE Connector Architecture.
- Chapter 4, "Programming Tasks," describes programming tasks for resource adapters.
- Chapter 5, "Connection Management," introduces you to resource adapter connection management.
- Chapter 6, "Transaction Management," introduces you to the resource adapter transaction management.
- Chapter 7, "Message and Transactional Inflow," describes resource adapter messaging inflow and transactional inflow.
- Chapter 8, "Security," describes how to configure security for resource adapters.
- Chapter 9, "Packaging and Deploying Resource Adapters," discusses packaging and deploying requirements for resource adapters and provides instructions for performing these tasks.

- Appendix A, “weblogic-ra.xml Schema,” provides a complete reference for the schema for the WebLogic Server-specific deployment descriptor, `weblogic-ra.xml`.
- Appendix B, “Resource Adapter Best Practices,” provides best practices for resource adapter developers.

Related Documentation

The foundation document for resource adapter development is Sun Microsystems J2EE Connector Architecture Specification, Version 1.5 Final Release (referred to in this guide as the J2CA 1.5 Specification): http://java.sun.com/j2ee/connector/ . This document assumes you are familiar with the J2CA 1.5 Specification and contains design and development information that is specific to developing WebLogic Server resource adapters.

For comprehensive guidelines for developing, deploying, and monitoring WebLogic Server applications, see the following documents:

- *Developing Applications with WebLogic Server* is a guide to developing WebLogic Server applications.
- *Deploying Applications to WebLogic Server*is the primary source of information about deploying WebLogic Server applications.
- *WebLogic Server Performance and Tuning* contains information on monitoring and improving the performance of WebLogic Server applications.

Examples for the Resource Adapter Developer

In addition to this document, BEA Systems provides resource adapter examples for software developers. WebLogic Server optionally installs API code examples in `WL_HOME\samples\server\examples\src\examples`, where `WL_HOME` is the top-level directory of your WebLogic Server installation. You can start the examples server, and obtain information about the samples and how to run them from the WebLogic Server Start menu.

The resource adapter examples provided with this release of WebLogic Server are compliant with the J2CA 1.5 Specification. BEA recommends that you examine, run, and understand these resource adapter examples before developing your own resource adapters.

New and Changed Features in This Release

For information about new, changed, and deprecated features in this release of WebLogic Server, see What's New in WebLogic Server 10.0 in *WebLogic Server Release Notes*.

Understanding Resource Adapters

The following sections introduce WebLogic resource adapters, the BEA WebLogic Server implementation of the J2EE Connector Architecture:

- "Overview of Resource Adapters" on page 2-1
- "J2EE Connector Architecture" on page 2-4
- "Resource Adapter Deployment Descriptors" on page 2-7

Overview of Resource Adapters

A resource adapter is a system library specific to an Enterprise Information System (EIS) and provides connectivity to an EIS; a resource adapter is analogous to a JDBC driver, which provides connectivity to a database management system. The interface between a resource adapter and the EIS is specific to the underlying EIS; it can be a native interface. The resource adapter plugs into an application server, such as WebLogic Server, and provides seamless connectivity between the EIS, application server, and enterprise application.

Multiple resource adapters can plug in to an application server. This capability enables application components deployed on the application server to access the underlying EISes. An application server and an EIS collaborate to keep all system-level mechanisms—transactions, security, and connection management—transparent to the application components. As a result, an application component provider can focus on the development of business and presentation logic for application components and need not get involved in the system-level issues related to EIS integration. This leads to an easier and faster cycle for the development of scalable, secure, and transactional enterprise applications that require connectivity with multiple EISes.

Comparing WebLogic Server and WebLogic Integration Resource Adapters

It is important to note the difference between BEA WebLogic Integration (WLI) resource adapters and BEA WebLogic Server resource adapters. WebLogic Integration resource adapters are written to be specific to WebLogic Server and, in general, are not deployable to other application servers. However, WebLogic Server resource adapters written without WLI extensions are deployable in any J2EE-compliant application server. This document discusses the design and implementation of non-WLI resource adapters. For more information on WebLogic Integration resource adapters, see *BEA WebLogic Adapter 8.1 Documentation*.

Inbound, Outbound, and Bidirectional Resource Adapters

WebLogic Server supports three types of resource adapters:

- Outbound resource adapter (supported by J2EE 1.0 and 1.5 Connector Architecture)—Allows an application to connect to an EIS system and perform work. All communication is initiated by the application. In this case, the resource adapter serves as a passive library for connecting to an EIS and executes in the context of the application threads.

  Outbound resource adapters based on the J2EE 1.5 Connector Architecture can be configured to have more than one simultaneous outbound connection. You can configure each outbound connection to have its own WebLogic Server-specific authentication and transaction support. See "Configuring Outbound Connections" on page 5-4.

  Outbound resource adapters based on the J2EE 1.0 Connector Architecture are also supported. These resource adapters can have only one outbound connection.

- Inbound resource adapter (1.5 only)—Allows an EIS to call application components and perform work. All communication is initiated by the EIS. The resource adapter may request threads from WebLogic Server or create its own threads; however, this is not the BEA-recommended approach. BEA recommends that the resource adapter submit work by way of the WorkManager. See Chapter 7, "Message and Transactional Inflow."

  Note: The WebLogic Server thin-client JAR also supports the WorkManager contracts and thus can be used by non-managed resource adapters (resource adapters not running in WebLogic Server.)

- Bi-directional resource adapter (1.5 only)—Supports both outbound and inbound communication.

Comparing 1.0 and 1.5 Resource Adapters

WebLogic Server supports resource adapters developed under either the J2EE 1.0 Connector Architecture or the J2EE 1.5 Connector Architecture. The J2EE 1.0 Connector Architecture restricts resource adapter communication to a single external system using one-way outbound communication. The J2EE 1.5 Connector Architecture lifts this restriction. Other capabilities provided by and for a 1.5 resource adapter that do not apply to 1.0 resource adapters include:

- Outbound communication from the application to multiple systems, such as Enterprise Information Systems (EISes) and databases. See "Inbound, Outbound, and Bidirectional Resource Adapters" on page 2-2.
- Inbound communication from one or more external systems such as an EIS to the resource adapter. See "Handling Inbound Messages" on page 7-5
- Transactional inflow to enable a J2EE application server to participate in an XA transaction controlled by an external Transaction Manager by way of a resource adapter. See "Transactional Inflow" on page 7-9.
- An application server-provided Work Manager to enable resource adapters to create threads through `Work` instances. See "work-manager" on page A-6.
- A life cycle contract for calling `start()` and `stop()` methods of the resource adapter by the application server. See "Programming a Resource Adapter to Perform as a Startup Class" on page 4-2.

Another important difference between 1.0 resource adapters and 1.5 resource adapters has to do with connection pools. For 1.5 resource adapters, you do not automatically get one connection pool per connection factory; you must configure a connection instance. You do so by setting the `connection-instance` element in the `weblogic-ra.xml` deployment descriptor.

Although WebLogic Server is now compliant with the J2EE 1.5 Connector Architecture, it continues to fully support the J2EE 1.0 Connector Architecture. In accordance with the J2EE 1.5 Connector Architecture, WebLogic Server now supports schema-based deployment descriptors. Resource adapters that have been developed based on the J2EE 1.0 Connector Architecture use Document Type Definition (DTD)-based deployment descriptors. Resource adapters that are built on DTD-based deployment descriptors are still supported.

This document describes the development and use of 1.5 resource adapters. For more information on WebLogic Server resources adapters that are based on the J2EE 1.0 Connector Architecture, see the BEA WebLogic Server 8.1 version of *Programming WebLogic Resource Adapters*.

J2EE Connector Architecture

The J2EE Connector Architecture defines a standard architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems (EISes), such as Enterprise Resource Planning (ERP) systems, mainframe transaction processing (TP), and database systems

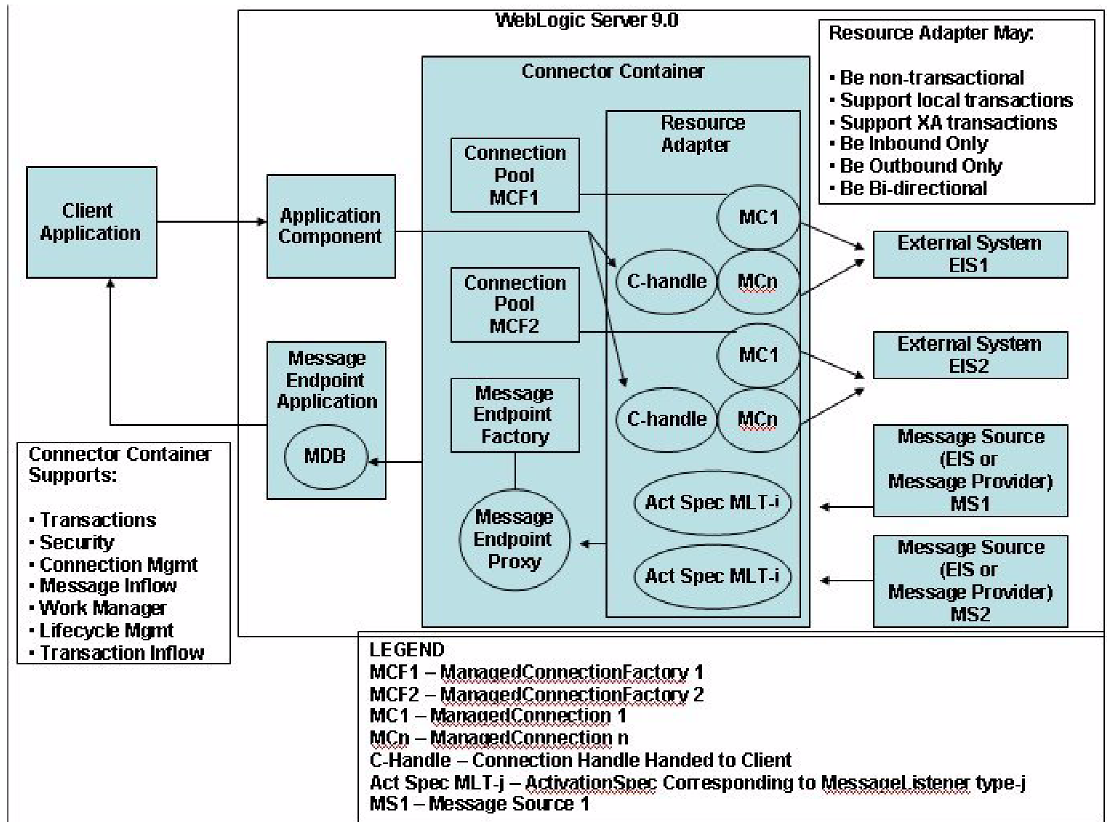
The resource adapter serves as a protocol adapter that allows any arbitrary EIS communication protocol to be used for connectivity. An application server vendor extends its system once to support the J2EE Connector Architecture and is then assured of seamless connectivity to multiple EISes. Likewise, an EIS vendor provides one standard resource adapter that can plug in to any application server that supports the J2EE Connector Architecture.

For a more detailed overview of the J2EE Connector Architecture, see Section 3 “The Connector Architecture” of the J2CA 1.5 Specification.

J2EE Architecture Diagram and Components

Figure 2-1 and the discussion that follows describe a WebLogic Server implementation of the J2EE 1.5 Connector Architecture.

Figure 2-1 Connector Architecture Overview



The connector architecture shown in Figure 2-1, "Connector Architecture Overview," on page 2-5 demonstrates a bi-directional resource adapter. The following components are used in outbound connection operations:

- A client application that connects to WebLogic Server, but also needs to interact with the EIS.
- An application component (an EJB or servlet) that the client application uses to submit outbound requests to the EIS through the resource adapter
- The WebLogic Server Connector container in which the resource adapter is deployed. The container in this example holds the following:

  - A deployed resource adapter that provides bi-directional (inbound and outbound) communication to and from the EIS.
  - One or more connection pools maintained by the container for the management of outbound managed connections for a given `ManagedConnectionFactory` (in this case, MCF-2—there may be more corresponding to different types of connections to a single EIS or even different EISes)
  - Multiple managed connections (MC1, MCn), which are objects representing the outbound physical connections from the resource adapter to the EIS.
  - Connection handles (C-handle) returned to the application component from the connection factory of the resource adapter and used by the application component for communicating with the EIS.

The following components are used for inbound connection operations:

- One or more external message sources (MS1, MS2), which could be an Enterprise Information System (EIS) or Message Provider, and which send messages inbound to WebLogic Server.
- One or more ActivationSpecs (Act Spec), each of which corresponds to a single MessageListener type (MLT-i).
- A `MessageEndpointFactory` created by the EJB container and used by the resource adapter for inbound messaging to create proxies to `MessageEndpoint` instances (MDB instances from the MDB pool).
- A message endpoint application (a message-driven bean (MDB) and possibly other J2EE components) that receives and handles inbound messages from the EIS through the resource adapter.

System-Level Contracts

To achieve a standard system-level pluggability between WebLogic Server and an EIS, WebLogic Server has implemented the standard set of system-level contracts defined by the J2EE Connector Architecture. These contracts consist of SPI classes and interfaces that are required to be implemented in the application server and the EIS, so that the two systems can work cooperatively. The EIS side of these system-level contracts are implemented in the resource adapter's Java classes. The following standard contracts are supported:

- Connection management contract—Enables WebLogic Server to pool connections to an underlying EIS and enables application components to connect to an EIS. Also allows

efficient use of connection resources through resource sharing and provides controls for associating and disassociating connection handles with EIS connections.

- Transaction management contract—A contract between the transaction manager and an EIS that supports transactional access to EIS resource managers. Enables WebLogic Server to use its transaction manager to manage transactions across multiple resource managers.
- Transaction inflow contract—Allows a resource adapter to propagate an imported transaction to WebLogic Server. Allows a resource adapter to flow-in transaction completion and crash recovery calls initiated by an EIS. Transaction inflow involves the use of an external transaction manager to coordinate transactions.
- Security contract—Provides secure access to an EIS and support for a secure application environment that reduces security threats to the EIS and protects valuable information resources managed by the EIS.
- Life cycle management contract—Enables WebLogic Server to manage the life cycle of a resource adapter. This allows bootstrapping a resource adapter instance during its deployment or application server startup, and notification to the resource adapter instance when it is undeployed or when the application server is being shut down.
- Work management contract—Allows a resource adapter to do work (monitor network endpoints, call application components, and so on) by submitting `Work` instances to WebLogic Server for execution.
- Message inflow contract—Allows a resource adapter to asynchronously or synchronously deliver messages to message endpoints residing in WebLogic Server independent of the specific messaging style, messaging semantics, and messaging infrastructure used to deliver messages. Also serves as the standard message provider pluggability contract that enables a wide range of message providers (Java Message Service, Java API for XML Messaging, and so on) to be plugged into WebLogic Server through a resource adapter.

These system-level contracts are described in detail in the J2CA 1.5 Specification.

Resource Adapter Deployment Descriptors

The structure of a resource adapter and its run-time behavior are defined in deployment descriptors. Programmers create the deployment descriptors during the packaging process, and these become part of the application deployment when the application is compiled.

WebLogic Server resource adapters have two deployment descriptors, each of which has its own XML schema:

- `ra.xml`—The standard J2EE deployment descriptor. All resource adapters must be specified in an `ra.xml` deployment descriptor file. The schema for `ra.xml` is `http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd`.
- `weblogic-ra.xml`—This WebLogic Server-specific deployment descriptor contains elements related to WebLogic Server features such as transaction management, connection management, and security. This file is required for the resource adapter to be deployed to WebLogic Server. The schema for the `weblogic-ra.xml` deployment descriptor file is http://www.bea.com/ns/weblogic/90/weblogic-ra.xsd. For a reference to the `weblogic-ra.xml` deployment descriptor, see Appendix A, “weblogic-ra.xml Schema.”

Creating and Configuring Resource Adapters

The following sections describe how to create and configure a WebLogic Server resource adapter:

- "Creating and Configuring Resource Adapters: Main Steps" on page 3-1
- "Modifying an Existing Resource Adapter" on page 3-3
- "Configuring the ra.xml File" on page 3-4
- "Configuring the weblogic-ra.xml File" on page 3-4

Creating and Configuring Resource Adapters: Main Steps

This section describes how to create a new WebLogic resource adapter. The next section, "Modifying an Existing Resource Adapter" on page 3-3, describes how to take an existing resource adapter and prepare it for deployment on WebLogic Server.

To create a new WebLogic resource adapter, you must create the classes for the particular resource adapter (`ConnectionFactory`, `Connection`, and so on), write the resource adapter's deployment descriptors, and then package everything into an archive file to be deployed to WebLogic Server.

The following are the main steps for creating a resource adapter:

1. Write the Java code for the various classes required by resource adapter (`ConnectionFactory`, `Connection`, and so on) in accordance with the J2CA 1.5 Specification. You will specify these 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>`

For more information, see Chapter 4, “Programming Tasks.”

2. Compile the Java code for the interfaces and implementation into class files, using a standard compiler.

3. Create the resource adapter’s deployment descriptors. A WebLogic resource adapter uses two deployment descriptor files:

   - `ra.xml` describes the resource adapter-related attributes type and its deployment properties using the standard XML schema specified by the J2CA 1.5 Specification.
   - `weblogic-ra.xml` adds additional WebLogic Server-specific deployment information, including connection and connection pool properties, security identities, Work Manager properties, and logging.

   For detailed information about creating WebLogic Server-specific deployment descriptors for resource adapters, refer to “Configuring the weblogic-ra.xml File” on page 3-4 and Appendix A, “weblogic-ra.xml Schema.”

4. Package the Java classes into a Java archive (JAR) file with a `.rar` extension.

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

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

5. Deploy the resource adapter archive (RAR) file on WebLogic Server in a test environment and test it.

   During testing, you may need to edit the resource adapter deployment descriptors. You can do this using the WebLogic Server Administration Console or manually using an XML editor or a text editor. For more information about editing deployment descriptors, see “Configuring the weblogic-ra.xml File” on page 3-4 and Configure resource adapter

properties in the Administration Console online help. See also Appendix A, "weblogic-ra.xml Schema" for detailed information on the elements in the deployment descriptor.

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.

   For information about these steps, see Chapter 9, "Packaging and Deploying Resource Adapters." See also *Deploying WebLogic Server Applications* for detailed information about deploying components and applications in general.

Modifying an Existing Resource Adapter

In some cases, you may already have a resource adapter available for deployment to WebLogic Server. This section describes how to take an existing resource adapter that is packaged in a RAR file and modify it for deployment to WebLogic Server. This involves adding the `weblogic-ra.xml` deployment descriptor and repackaging the resource adapter. The following procedure supposes you have an existing resource adapter packaged in a RAR file named `blackbox-notx.rar`.

1. Create a temporary directory anywhere on your hard drive to stage the resource adapter:

```
mkdir c:/stagedir
```

2. 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 `Manifest.mf` and `ra.xml` files

   Execute these commands to see these files:

```
c:/stagedir> ls

       blackbox-notx.rar

       META-INF

c:/stagedir> ls META-INF

       Manifest.mf
```

```
ra.xml
```

3. 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 settings.

   For more information, see "Configuring the weblogic-ra.xml File" on page 3-4 and also refer to Appendix A, "weblogic-ra.xml Schema," for information on the XML schema that applies to `weblogic-ra.xml`.

4. 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
   ```

5. Create the resource adapter archive:

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

6. Deploy the resource adapter to WebLogic Server. For more information about packaging and deploying the resource adapter, see Chapter 9, "Packaging and Deploying Resource Adapters" and *Deploying Applications to WebLogic Server*.

Configuring the ra.xml File

If your resource adapter does not already contain a `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 or XML editor to edit the properties. For information on creating a `ra.xml` file, refer to the J2CA 1.5 Specification.

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 `weblogic-*.xml` extensions for EJBs and Web applications in WebLogic Server, which also add WebLogic-specific deployment descriptors to the deployable archive. The basic RAR or deployment directory can be deployed to WebLogic Server without a `weblogic-ra.xml` file. 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. However, this automatically generated `weblogic-ra.xml` file is not persisted to the RAR; the RAR remains unchanged.

The following summarizes the most significant features you can configure in the `weblogic-ra.xml` deployment descriptor file.

- Descriptive text about the connection factory.
- JNDI name bound to a connection factory. (Resource adapters developed based on the J2CA 1.5 Specification are bound in the JNDI as objects independently of their `ConnectionFactory` objects.)
- Reference to a separately deployed connection factory that contains resource adapter components that can be shared with the current resource adapter.
- Connection pool parameters that set the following behavior:
  - Initial number of `ManagedConnections` that WebLogic Server attempts to allocate at deployment time.
  - Maximum number of `ManagedConnections` that WebLogic Server allows to be allocated at any one time.
  - Number of `ManagedConnections` that WebLogic Server attempts to allocate when filling a request for a new connection.
  - Whether WebLogic Server attempts to reclaim unused `ManagedConnections` to save system resources.
  - The time WebLogic Server waits between attempts to reclaim unused `ManagedConnections`.
- Logging properties to configure WebLogic Server logging for the `ManagedConnectionFactory` or `ManagedConnection`.
- Transaction support levels (XA, local, or no transaction support).
- Principal names to use as security identities.

For detailed information about configuring the weblogic-ra.xml deployment descriptor file, see the reference information in Appendix A, "weblogic-ra.xml Schema." See also the configuration information in the following sections:

- Chapter 5, "Connection Management"
- Chapter 6, "Transaction Management"
- Chapter 7, "Message and Transactional Inflow"
- Chapter 8, "Security"

Editing Resource Adapter Deployment Descriptors

To define or make changes to the XML descriptors used in the WebLogic Server resource adapter archive, you must define or edit the XML elements in the weblogic-ra.xml and ra.xml deployment descriptor files. You can edit the deployment descriptor files with any plain text editor. However, to avoid introducing errors, use a tool designed for XML editing.You can also edit most elements of the files using the WebLogic Administration Console.

Editing Considerations

To edit XML elements manually:

- If you use an ASCII text editor, make sure that it 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>`

Schema Header Information

When editing or creating XML deployment files, it is critical to include the correct schema header for each deployment file. The header refers to the location and version of the schema for the deployment descriptor.

Although this header references an external URL at java.sun.com, WebLogic Server contains its own copy of the schema, so your host server need not have access to the Internet. However, you must still include this `<?xml version...>` element in your ra.xml file, and have it

reference the external URL because the version of the schema contained in this element is used to identify the version of this deployment descriptor.

Table 3-1 shows the entire schema headers for the `ra.xml` and `weblogic-ra.xml` files.

Table 3-1 Schema Header

XML File	Schema Header

ra.xml	`<?xml version="1.0" encoding="UTF-8"?>`
	`<connector xmlns="http://java.sun.com/xml/ns/j2ee"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee`
	`http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd"`
	`version="1.5">`

weblogic-ra.xml	`<?xml version = "1.0">`
	`<weblogic-connector`
	`xmlns="http://www.bea.com/ns/weblogic/90">`

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'
```

Conforming Deployment Descriptor Files to Schema

The contents and arrangement of elements in your deployment descriptor files must conform to the schema for each file you use. The following links provide the public schema locations for deployment descriptor files used with WebLogic Server:

- `connector_1_5.xsd` contains the schema for the standard `ra.xml` deployment file, required for all resource adapters. This schema is maintained as part of the J2CA 1.5 Specification. It is located at: `http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd`
- `weblogic-ra.xsd` contains the schema used for creating `weblogic-ra.xml`, which defines resource adapter properties used for deployment to WebLogic Server. This schema is located at `http://www.bea.com/ns/weblogic/90/weblogic-ra.xsd`

Note: Your browser might not display the contents of files having the `.xsd` extension. In that case, to view the schema contents in your browser, save the links as text files and view them with a text editor.

Dynamic Descriptor Updates: Console Configuration Tabs

You can use the Administration Console to view, modify, and (when necessary) persist deployment descriptor elements. Some descriptor element changes take place dynamically at runtime without requiring the resource adapter to be redeployed. Other descriptor elements require redeployment after changes are made. To use the Administration Console to configure a resource adapter, open Deployments and click the name of the deployed resource adapter. Use the Configuration tab to change the configuration of the resource adapter and the other tabs to control, test, or monitor the resource adapter. For information about using the Administration Console, see Configure Resource Adapter Properties in the console help.

Dynamic Pool Parameters

Using the Administration Console, you can modify the following `weblogic-ra.xml` pool parameters dynamically, without requiring the resource adapter to be redeployed:

- `initial-capacity`
- `max-capacity`
- `capacity-increment`
- `shrink-frequency-seconds`
- `highest-num-waiters`
- `highest-num-unavailable`
- `connection-creation-retry-frequency-seconds`
- `connection-reserve-timeout-seconds`
- `test-frequency-seconds`

Dynamic Logging Parameters

Using the Administration Console, you can modify the following `weblogic-ra.xml` logging parameters dynamically, without requiring the resource adapter to be redeployed:

- `log-filename`
- `file-count`
- `file-size-limit`
- `log-file-rotation-dir`
- `rotation-time`
- `file-time-span`

Automatic Generation of the weblogic-ra.xml File

A resource adapter archive (RAR) deployed on WebLogic Server must include a `weblogic-ra.xml` deployment descriptor file in addition to the `ra.xml` deployment descriptor file specified in the J2CA 1.5 Specification.

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. However, this automatically generated `weblogic-ra.xml` file is not persisted to the RAR; the RAR remains unchanged. WebLogic Server instead generates internal data structures that correspond to default information in the `weblogic-ra.xml` file.

For a 1.0 resource adapter that is a single connection factory definition, the JNDI name will be `eis/`*`ModuleName`*. For example, if the RAR is named `MySpecialRA.rar`, the JNDI name of the connection factory will be `eis/MySpecialRA`.

For a 1.5 resource adapter with a `ResourceAdapter` bean class specified, the JNDI name of the bean would be `MySpecialRA`. Each connection factory would also have a corresponding instance created with a JNDI name of `eis/`*`ModuleName`*, `eis/`*`ModuleName_1`*, `eis/`*`ModuleName_2`*, and so on.

(Deprecated) Configuring the Link-Ref Mechanism

The Link-Ref mechanism was introduced in the 8.1 release of WebLogic Server to enable the deployment of a single base adapter whose code could be shared by multiple logical adapters with various configuration properties. For 1.5 resource adapters in the current release, the Link-Ref mechanism is deprecated and is replaced by the new J2EE libraries feature. However, the Link-Ref mechanism is still supported in this release for 1.0 resource adapters. For more information on J2EE libraries, see "Creating Shared J2EE Libraries and Optional Packages" in *Developing Applications with WebLogic Server*. To use the Link-Ref mechanism, use the `<ra-link-ref>` element in your resource adapter's `weblogic-ra.xml` file.

The deprecated and 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). 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 further instructions on editing the `weblogic-ra.xml` file, see Appendix A, "weblogic-ra.xml Schema."

Programming Tasks

The following sections discuss programming tasks for WebLogic Server resource adapters:

- "Required Classes for Resource Adapters" on page 4-1
- "Programming a Resource Adapter to Perform as a Startup Class" on page 4-2
- "Suspending and Resuming Resource Adapter Activity" on page 4-7
- "Extended BootstrapContext" on page 4-13

Required Classes for Resource Adapters

A resource adapter requires the following Java classes, in accordance with the J2CA 1.5 Specification:

- `ManagedConnectionFactory`
- `ConnectionFactory` interface
- `ConnectionFactory` implementation
- `Connection` interface
- `Connection` implementation

These classes are specified in the `ra.xml` file. For example:

```
– <managedconnectionfactory-class>com.sun.connector.blackbox.LocalTxMan
  agedConnectionFactory</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>`

In addition, if the resource adapter supports inbound messaging, the resource adapter will require an `ActivationSpec` class for each supported inbound message type. See Chapter 7, "Message and Transactional Inflow."

The specifics of these resource adapter classes depend on the nature of the resource adapter you are developing.

Programming a Resource Adapter to Perform as a Startup Class

As an alternative to using a WebLogic Server startup class, you can program a resource adapter with a minimal resource adapter class that implements `javax.resource.ResourceAdapter`, which defines a `start()` and `stop()` method.

Note: Because of the definition of the `ResourceAdapter` interface, you must also define the `endpointActivation()`, `Deactivation()` and `getXAResource()` methods.

When the resource adapter is deployed, the `start()` method is invoked. When it is undeployed, the `stop()` method is called. Any work that the resource adapter initiates can be performed in the `start()` method as with a WebLogic Server startup class.

Because resource adapters have access to the Work Manager through the `BootstrapContext` in the `start()` method, they should submit Work instances instead of using direct thread management. This enables WebLogic Server to manage threads effectively through its self-tuning Work Manager.

Once a Work instance is submitted for execution, the `start()` method should return promptly so as not to interfere with the full deployment of the resource adapter. Thus, a `scheduleWork()` or `startWork()` method should be invoked on the Work Manager rather than the `doWork()` method.

The following is an example of a resource adapter having a minimum resource adapter class. It is the absolute minimum resource adapter that you can develop (other than removing the

println statements). In this example, the only work performed by the start() method is to print a message to stdout (standard out).

Listing 4-1 Minimum Resource Adapter

```
import javax.resource.spi.ResourceAdapter;
import javax.resource.spi.endpoint.MessageEndpointFactory;
import javax.resource.spi.ActivationSpec;
import javax.resource.ResourceException;
import javax.transaction.xa.XAResource;
import javax.resource.NotSupportedException;
import javax.resource.spi.BootstrapContext;
/**
* This resource adapter is the absolute minimal resource adapter that anyone
can build (other than removing the println's.)
*/
public class ResourceAdapterImpl implements ResourceAdapter
{
   public void start( BootstrapContext bsCtx )
   {
      System.out.println( "ResourceAdapterImpl started" );
   }
   public void stop()
   {
      System.out.println( "ResourceAdapterImpl stopped" );
   }
   public void endpointActivation(MessageEndpointFactory
messageendpointfactory, ActivationSpec activationspec)
      throws ResourceException
```

```
    {
       throw new NotSupportedException();
    }
    public void endpointDeactivation(MessageEndpointFactory
messageendpointfactory, ActivationSpec activationspec)
    {
    }
    public XAResource[] getXAResources(ActivationSpec aactivationspec[])
       throws ResourceException
    {
       throw new NotSupportedException();
    }
}
```

The following is an example of a resource adapter that submits work instances to the Work Manager. The resource adapter starts some work in the `start()` method, thus serving as a J2EE-compliant startup class.

Listing 4-2 Resource Adapter Using the Work Manager and Submitting Work Instances

```
import javax.resource.NotSupportedException;
import javax.resource.ResourceException;
import javax.resource.spi.ActivationSpec;
import javax.resource.spi.BootstrapContext;
import javax.resource.spi.ResourceAdapter;
import javax.resource.spi.endpoint.MessageEndpointFactory;
import javax.resource.spi.work.Work;
import javax.resource.spi.work.WorkException;
```

```
import javax.resource.spi.work.WorkManager;
import javax.transaction.xa.XAResource;
/**
* This Resource Adapter starts some work in the start() method, thus serving
as a J2EE compliant "startup class"
*/
public class ResourceAdapterWorker implements ResourceAdapter
{
   private WorkManager wm;
   private MyWork someWork;
   public void start( BootstrapContext bsCtx )
   {
      System.out.println( "ResourceAdapterWorker started" );
      wm = bsCtx.getWorkManager();
      try
      {
         someWork = new MyWork();
         wm.startWork( someWork );
      }
      catch (WorkException ex)
      {
         System.err.println( "Unable to start work: " + ex );
      }
   }
   public void stop()
   {
   // stop work that was started in the start() method
   someWork.release();
```

```
        System.out.println( "ResourceAdapterImpl stopped" );
    }
    public void endpointActivation(MessageEndpointFactory
messageendpointfactory, ActivationSpec activationspec)
        throws ResourceException
    {
        throw new NotSupportedException();
    }
    public void endpointDeactivation(MessageEndpointFactory
messageendpointfactory, ActivationSpec activationspec)
    {
    }
    public XAResource[] getXAResources(ActivationSpec activationspec[])
throws ResourceException
    {
        throw new NotSupportedException();
    }
    // Work class
    private class MyWork implements Work
    {
        private boolean isRunning;
        public void run()
        {
            isRunning = true;
            while (isRunning)
            {
        // do a unit of work (e.g. listen on a socket, wait for an inbound msg,
            // check the status of something)
```

```
            System.out.println( "Doing some work" );
            // perhaps wait some amount of time or for some event
            try
            {
               Thread.sleep( 60000 ); // wait a minute
            }
            catch (InterruptedException ex)
            {}
         }
      }
      public void release()
      {
            // signal the run() loop to stop
            isRunning = false;
         }
      }
}
```

Suspending and Resuming Resource Adapter Activity

You can program your resource adapter to use the `suspend()` method, which provides custom behavior for suspending activity. For example, using the `suspend()` method, you can queue up all incoming messages while allowing in-flight transactions to complete, or you can notify the Enterprise Information System (EIS) that reception of messages is temporarily blocked.

You then invoke the `resume()` method to signal that the inbound queue be drained and messages be delivered, or notify the EIS that message receipt was re-enabled. Basically, the `resume()` method allows the resource adapter to continue normal operations.

You initiate the `suspend()` and `resume()` methods by making a call on the resource adapter runtime MBeans programmatically, using WebLogic Scripting Tool, or from the WebLogic

Server Administration Console. See Start, stop, and suspend resource adapters in the console help for more information.

The `Suspendable.supportsSuspend()` method determines whether a resource adapter supports a particular type of suspension. The `Suspendable.isSuspended()` method determines whether or not a resource adapter is presently suspended.

A resource adapter that supports `suspend()`, `resume()`, or production redeployment must implement the `Suspendable` interface to inform WebLogic Server that these operations are supported. These operations are invoked by WebLogic Server when the following occurs:

- Suspend is called by the `suspend()` method on the connector component MBean.
- The production redeployment sequence of calls is invoked (when a new version of the application is deployed that contains the resource adapter). See "Suspendable Interface and Production Redeployment" on page 9-6.

Listing 4-3 contains the `Suspendable` interface for resource adapters:

Listing 4-3 Suspendable Interface

```
package weblogic.connector.extensions;
import java.util.Properties;
import javax.resource.ResourceException;
import javax.resource.spi.ResourceAdapter;
/**
* Suspendable may be implemented by a ResourceAdapter JavaBean if it
* supports suspend, resume or side-by-side versioning
* @author Copyright (c) 2002 by BEA Systems, Inc. All Rights Reserved.
* @since November 14, 2003
*/
public interface Suspendable
{
/**
* Used to indicate that inbound communication is to be suspended/resumed
```

```
*/
int INBOUND = 1;
/**
* Used to indicate that outbound communication is to be suspended/resumed
*/
int OUTBOUND = 2;
/**
* Used to indicate that submission of Work is to be suspended/resumed
*/
int WORK = 4;
/**
* Used to indicate that INBOUND, OUTBOUND & WORK are to be suspended/resumed
*/
int ALL = 7;
/**
* May be used to indicate a suspend() operation
*/
int SUSPEND = 1;
/**
* May be used to indicate a resume() operation
*/
int RESUME = 2;
/**
* Request to suspend the activity specified. The properties may be null or
* specified according to RA-specific needs
* @param type An int from 1 to 7 specifying the type of suspension being
* requested (i.e. Suspendable.INBOUND, .OUTBOUND, .WORK or the sum of one
```

```
* or more of these, or the value Suspendable.ALL )
* @param props Optional Properties (or null) to be used for ResourceAdapter
* specific purposes
* @exception ResourceException If the resource adapter can't complete the
* request
*/
void suspend( int type, Properties props ) throws ResourceException;
/**
* Request to resume the activity specified. The Properties may be null or
* specified according to RA-specific needs
*
* @param type An int from 1 to 7 specifying the type of resume being
* requested (i.e. Suspendable.INBOUND, .OUTBOUND, .WORK or the sum of
* one or more of these, or the value Suspendable.ALL )
* @param props Optional Properties (or null) to be used for ResourceAdapter
* specific purposes
* @exception ResourceException If the resource adapter can't complete the
* request
*/
void resume( int type, Properties props ) throws ResourceException;
/**
*
* @param type An int from 1 to 7 specifying the type of suspend this inquiry
* is about (i.e. Suspendable.INBOUND, .OUTBOUND, .WORK or the sum of
* one or more of these, or the value Suspendable.ALL )
* @return true iff the specified type of suspend is supported
*/
```

```
boolean supportsSuspend( int type );
/**
*
* Used to determine whether the specified type of activity is currently
suspended.
*
* @param type An int from 1 to 7 specifying the type of activity
* requested (i.e. Suspendable.INBOUND, .OUTBOUND, .WORK or the sum of
* one or more of these, or the value Suspendable.ALL )
* @return true iff the specified type of activity is suspened by this
resource adapter
*/
boolean isSuspended( int type );
/**
* Used to determine if this resource adapter supports the init() method used
for
* resource adapter versioning (side-by-side deployment)
*
* @return true iff this resource adapter supports the init() method
*/
boolean supportsInit();
/**
* Used to determine if this resource adapter supports the startVersioning()
method used for
* resource adapter versioning (side-by-side deployment)
*
* @return true iff this resource adapter supports the startVersioning()
method
*/
```

```
boolean supportsVersioning();
/**
* Used by WLS to indicate to the current version of this resource adapter
that a new
* version of the resource adapter is being deployed. This method can be used
by the
* old RA to communicate with the new RA and migrate services from the old
to the new.
* After being called, the ResourceAdapter is responsible for notifying the
Connector
* container via the ExtendedBootstrapContext.complete() method, that it is
safe to
* be undeployed.
*
* @param ra The new ResourceAdapter JavaBean
* @param props Properties associated with the versioning
* when it can be undeployed
* @exception ResourceException If something goes wrong
*/
void startVersioning( ResourceAdapter ra,
Properties props ) throws ResourceException;
/**
* Used by WLS to inform a ResourceAdapter that it is a new version of an
already deployed
* resource adapter. This method is called prior to start() so that the new
resource adapter
* may coordinate its startup with the resource adapter it is replacing.
*
* @param ra The old version of the resource adapter that is currently running
```

```
* @param props Properties associated with the versioning operation
* @exception ResourceException If the init() fails.
*/
void init( ResourceAdapter ra, Properties props ) throws ResourceException;
}
```

Extended BootstrapContext

If, when a resource adapter is deployed, it has a resource adapter JavaBean specified in the `<resource-adapter-class>` element of its `ra.xml` descriptor, the WebLogic Server connector container calls the `start()` method on the resource adapter bean as required by the J2CA 1.5 Specification. The resource adapter code can use the `BootstrapContext` object that is passed in by the `start()` method to:

- Obtain a `WorkManager` object for submitting `Work` instances
- Create a `Timer`
- Obtain an `XATerminator` for use in transaction inflow

These capabilities are all prescribed by the J2CA 1.5 Specification.

In addition to implementing the required `javax.resource.spi.BootstrapContext`, the `BootstrapContext` object passed to the resource adapter `start()` method also implements `weblogic.connector.extensions.ExtendedBootstrapContext`, which gives the resource adapter access to some additional WebLogic Server-specific extensions that enhance diagnostic capabilities. These extensions are described in the following sections.

Diagnostic Context ID

In the WebLogic Server diagnostic framework, a thread may have an associated *diagnostic context*. A request on the thread carries its diagnostic context throughout its lifetime, as it proceeds along its path of execution. The `ExtendedBootstrapContext` allows the resource adapter developer to set a diagnostic context *payload* consisting of a String that can be used, for example, to trace the execution of a request from an EIS all the way to a message endpoint. This capability can serve a variety of diagnostic purposes. For example, you can set the String to the client ID or session ID on an inbound message from an EIS. During message dispatch, various

diagnostics can be gathered to show the request flow through the system. As you develop your resource adapter classes, you can make use of the `setDiagnosticContextID()` and `getDiagnosticContextID()` methods for this purpose. For more information about the diagnostic context, see *Configuring and Using the WebLogic Diagnostic Framework*.

Dye Bits

The WebLogic Server diagnostic framework also provides the ability to *dye* a request. The `ExtendedBootstrapContext` allows you to set and retrieve four dye bits on the current thread for whatever diagnostic purpose the resource adapter developer chooses. For example, you might set priority of a request using the dye bits. For more information about request dyeing, see *Configuring and Using the WebLogic Diagnostic Framework*.

Callback Capabilities

You can use the `ExtendedBootstrapContext.complete()` method as a callback to the connector container. For detailed information on this feature, see "Production Redeployment" in *Deploying WebLogic Server Applications*.

CHAPTER 5

Connection Management

The following sections describe connection management in WebLogic Server resource adapters. For more information on connection management, see Chapter 6, "Connection Management," of the J2CA 1.5 Specification.

- "Connection Management Contract" on page 5-1
- "Configuring Outbound Connections" on page 5-4
- "Configuring Inbound Connections" on page 5-9
- "Configuring Connection Pool Parameters" on page 5-11
- "Connection Proxy Wrapper - 1.0 Resource Adapters" on page 5-15
- "Testing Connections" on page 5-16

Connection Management Contract

One of the requirements of the J2CA 1.5 Specification is the connection management contract. The connection management contract between WebLogic Server and a resource adapter:

- Provides a consistent application programming model for connection acquisition for both managed and non-managed (two-tier) applications.
- Enables a resource adapter to provide a connection factory and connection interfaces based on the common client interface (CCI) specific to the type of resource adapter and EIS. This enables JDBC drivers to be aligned with the J2EE 1.5 Connector Architecture with minimum impact on the existing JDBC APIs.

- Enables an application server to provide various services—transactions, security, advanced pooling, error tracing/logging—for its configured set of resource adapters.
- Supports connection pooling.

The resource adapter's side of the connection management contract is embodied in the resource adapter's `Connection`, `ConnectionFactory`, `ManagedConnection`, and `ManagedConnectionFactory` classes.

Connection Factory and Connection

A J2EE application component uses a public interface called a connection factory to access a connection instance, which the component then uses to connect to the underlying EIS. Examples of connections include database connections and JMS (Java Message Service) connections.

A resource adapter provides connection and connection factory interfaces, acting as a connection factory for EIS connections. For example, the`javax.sql.DataSource` and `java.sql.Connection` interfaces are JDBC-based interfaces for connecting to a relational database.

An application looks up a connection factory instance in the Java Naming and Directory Interface (JNDI) namespace and uses it to obtain EIS connections. See "Obtaining the ConnectionFactory (Client-JNDI Interaction)" on page 5-2.

Resource Adapters Bound in JNDI Tree

Version 1.5 resource adapters can be bound in the JNDI tree as independent objects, making them available as system resources in their own right or as message sources for message-driven beans (MDBs). In contrast, version 1.0 resource adapters are identified by their `ConnectionFactory` objects bound in the JNDI tree.

In a version 1.5 resource adapter, at deployment time, the `ResourceAdapter` Bean (if it exists) is bound into the JNDI tree using the value of the jndi-name element in the `weblogic-ra.xml` file. As a result, administrators can view resource adapters as single deployable entities, and they can interact with resource adapter capabilities publicly exposed by the resource adapter provider. For more information, see "jndi-name" on page A-2 in Appendix A, "weblogic-ra.xml Schema."

Obtaining the ConnectionFactory (Client-JNDI Interaction)

The application assembler or component provider configures the Connection Factory requirements for an application component in the application's deployment descriptor. For example:

```
res-ref-name: eis/myEIS
res-type: javax.resource.cci.ConnectionFactory
res-auth: Application or Container
```

The resource adapter deployer provides the configuration information for the resource adapter.

An application looks up a `ConnectionFactory` instance in the Java Naming and Directory Interface (JNDI) namespace and uses it to obtain EIS connections. The following events occur when an application in a managed environment obtains a connection to an EIS instance from a Connection Factory, as specified in the `res-type` variable.

Note: A managed application environment defines an operational environment for a J2EE-based, multi-tier, Web-enabled application that accesses EISes.

1. The application server uses a configured resource adapter to create physical connections to the underlying EIS.
2. The application component looks up a `ConnectionFactory` instance in the component's environment by using the JNDI interface, as shown in Listing 5-1.

Listing 5-1 JNDI Lookup

```
//obtain the initial JNDI Naming context
Context initctx = 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 an instance of type `java.resource.cci.ConnectionFactory` as specified in the `res-type` element.

3. The application component invokes the `getConnection` method on the `ConnectionFactory` 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();
   ```

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

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

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

   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.

Configuring Outbound Connections

Outbound resource adapters based on the J2CA 1.5 Specification can be configured to have one or more outbound connections, each having its own WebLogic Server-specific authentication and transaction support. You configure outbound connection properties in the `ra.xml` and `weblogic-ra.xml` deployment descriptor files.

Connection Pool Configuration Levels

You use the `outbound-resource-adapter` element and its subelements in the `weblogic-ra.xml` deployment descriptor to describe the outbound components of a resource adapter.

You can define outbound connection pools at three levels:

- Global—Specify parameters that apply to all outbound connection groups in the resource adapter using the `default-connection-properties` element. See "default-connection-properties" on page A-19.

- Group—Specify parameters that apply to all outbound connection instances belonging to a particular connection factory specified in the `ra.xml` deployment descriptor using the `connection-definition-group` element. A one-to-one correspondence exists from a connection factory in `ra.xml` to a connection definition group in `weblogic-ra.xml`. The properties specified in a group override any parameters specified at the global level. See "connection-definition-group" on page A-28.

The `connection-factory-interface` element (a subelement of the `connection-definition-group` element) serves as a required unique element (a key) to each `connection-definition-group`. There must be a one-to-one relationship between the `connection-definition-interface` element in `weblogic-ra.xml` and the `connectiondefinition-interface` element in `ra.xml`.

- Instance—Under each connection definition group, you can specify connection instances using the `connection-instance` element of the `weblogic-ra.xml` deployment descriptor. These correspond to the individual connection pools for the resource adapter. You can use the `connection-properties` subelement to specify properties at the instance level too; properties specified at the instance level override those provided at the group and global levels. See “connection-instance” on page A-29.

Multiple Outbound Connections Example

The following is an example of a `weblogic-ra.xml` deployment descriptor that configures multiple outbound connections:

Listing 5-2 weblogic-ra.xml Deployment Descriptor: Multiple Outbound Connections

```
<?xml version="1.0" ?>
<weblogic-connector xmlns="http://www.bea.com/ns/weblogic/90">
<jndi-name>900eisaNameOfBlackBoxXATx</jndi-name>
   <outbound-resource-adapter>
      <connection-definition-group>
         <connection-factory-interface>javax.sql.DataSource
         </connection-factory-interface>
            <connection-instance>
               <jndi-name>eis/900eisaBlackBoxXATxConnectorJNDINAME1
               </jndi-name>
               <connection-properties>
                  <pool-params>
                     <initial-capacity>2</initial-capacity>
                     <max-capacity>10</max-capacity>
```

```
                <capacity-increment>1</capacity-increment>
                <shrinking-enabled>true</shrinking-enabled>
               <shrink-frequency-seconds>60</shrink-frequency-seconds>
              </pool-params>
              <properties>
                <property>
                  <name>ConnectionURL</name>

<value>jdbc:oracle:thin:@bcpdb:1531:bay920;create=true;autocommit=false</v
alue>
                </property>
                <property>
                  <name>XADataSourceName</name>
                  <value>OracleXAPool</value>
                </property>
                <property>
                  <name>TestClassPath</name>
                  <value>HelloFromsetTestClassPathGoodDay</value>
                </property>
                <property>
                  <name>unique_ra_id</name>
                  <value>eisablackbox-xa.oracle.900</value>
                </property>
              </properties>
            </connection-properties>
          </connection-instance>
          <connection-instance>
              <jndi-name>eis/900eisaBlackBoxXATxConnectorJNDINAME2
```

```
                </jndi-name>
                <connection-properties>
                    <pool-params>
                        <initial-capacity>2</initial-capacity>
                        <max-capacity>10</max-capacity>
                        <capacity-increment>1</capacity-increment>
                        <shrinking-enabled>true</shrinking-enabled>
                        <shrink-frequency-seconds>60
                        </shrink-frequency-seconds>
                    </pool-params>
                <properties>
                    <property>
                        <name>ConnectionURL</name>

<value>jdbc:oracle:thin:@bcpdb:1531:bay920;create=true;autocommit=false</v
alue>
                    </property>
                    <property>
                        <name>XADataSourceName</name>
                        <value>OracleXAPool</value>
                    </property>
                    <property>
                        <name>TestClassPath</name>
                        <value>HelloFromsetTestClassPathGoodDay</value>
                    </property>
                    <property>
                        <name>unique_ra_id</name>
                        <value>eisablackbox-xa.oracle.900</value>
```

```
                        </property>
                     </properties>
                  </connection-properties>
               </connection-instance>
         </connection-definition-group>
        <connection-definition-group>
            <connection-factory-interface>javax.sql.DataSourceCopy
            </connection-factory-interface>
            <connection-instance>
             <jndi-name>eis/900eisaBlackBoxXATxConnectorJNDINAME3</jndi-name>
               <connection-properties>
                     <pool-params>
                           <initial-capacity>2</initial-capacity>
                           <max-capacity>10</max-capacity>
                           <capacity-increment>1</capacity-increment>
                           <shrinking-enabled>true</shrinking-enabled>
                      <shrink-frequency-seconds>60</shrink-frequency-seconds>
                     </pool-params>
                     <properties>
                        <property>
                           <name>ConnectionURL</name>

    <value>jdbc:oracle:thin:@bcpdb:1531:bay920;create=true;autocommit=false</v
    alue>
                        </property>
                        <property>
                           <name>XADataSourceName</name>
                           <value>OracleXAPoolB</value>
```

```
                    </property>
                    <property>
                        <name>TestClassPath</name>
                        <value>HelloFromsetTestClassPathGoodDay</value>
                    </property>
                    <property>
                        <name>unique_ra_id</name>
                        <value>eisablackbox-xa-two.oracle.900</value>
                    </property>
                </properties>
            </connection-properties>
        </connection-instance>
    </connection-definition-group>
  </outbound-resource-adapter>
</weblogic-connector>
```

Configuring Inbound Connections

The J2CA 1.5 Specification permits you to configure a resource adapter to support inbound message connections. The following are the main steps for configuring an inbound connection:

1. Provide a JNDI name for the resource adapter in the `weblogic-ra.xml` deployment descriptor. See "jndi-name" on page A-2.
2. Configure a message listener and `ActivationSpec` for each supported inbound message type in the `ra.xml` deployment descriptor. For information about requirements for an `ActivationSpec` class, see Chapter 12, "Message Inflow" in the J2CA 1.5 Specification.
3. Within the packaged enterprise application, include a configured EJB message-driven bean (MDB). In the `resource-adapter-jndi-name` element of the `weblogic-ejb-jar.xml` deployment descriptor, provide the same JNDI name assigned to the resource adapter in the previous step. Setting this value enables the MDB and resource adapter to communicate with each other.

4. Configure the security identity to be used by the resource adapter for inbound connections. When messages are received by the resource adapter, work must be performed under a particular security identity. See "Configuring Security Identities for Resource Adapters" on page 8-6.
5. Deploy the resource adapter as discussed in *Deploying WebLogic Server Applications*.
6. Deploy the MDB. For more information, see "Message-Driven EJBs" in *Programming WebLogic Enterprise JavaBeans* and *Deploying WebLogic Server Applications*.

The following example shows how an inbound connection with two message listener/activation specs could be configured in the `ra.xml` deployment descriptor:

Listing 5-3 Example of Configuring an Inbound Connection

```
<inbound-resourceadapter>
   <messageadapter>
      <messagelistener>
         <messagelistener-type>
         weblogic.qa.tests.connector.adapters.flex.InboundMsgListener
         </messagelistener-type>
         <activationspec>
            <activationspec-class>
            weblogic.qa.tests.connector.adapters.flex.ActivationSpecImpl
            </activationspec-class>
         </activationspec>
      </messagelistener>
      <messagelistener>
         <messagelistener-type>
       weblogic.qa.tests.connector.adapters.flex.ServiceRequestMsgListener
         </messagelistener-type>
         <activationspec>
```

```
            <activationspec-class>

weblogic.qa.tests.connector.adapters.flex.ServiceRequestActivationSpec
            </activationspec-class>
            </activationspec>
      </messagelistener>
   </messageadapter>
</inbound-resourceadapter>
```

Configuring Connection Pool Parameters

This section explains how to configure WebLogic Server resource adapter connection pool parameters in the `weblogic-ra.xml` deployment descriptor. For more details, see Appendix A, "weblogic-ra.xml Schema."

initial-capacity: Setting the Initial Number of ManagedConnections

Depending on the complexity of the Enterprise Information System (EIS) that the `ManagedConnection` is representing, creating `ManagedConnections` can be expensive. 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`.

Because no initiating security principal or request context information is known at WebLogic Server startup, you create initial connections using a security subject that you specify in the `security` element, as described in "Configuring Security Identities for Resource Adapters" on page 8-6.

max-capacity: Setting the Maximum Number of ManagedConnections

As more `ManagedConnections` are created, they consume more system resources—such as memory and disk space. Depending on the Enterprise Information System (EIS), this consumption may affect the performance of the overall system. To control the effects of `ManagedConnections` on system resources, you can specify a maximum number of allocated `ManagedConnections` in the `max-capacity` element of 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. Requests for newly allocated `ManagedConnections` beyond this limit results in a `ResourceAllocationException` being returned to the caller.

capacity-increment: Controlling the Number of ManagedConnections

In compliance with the J2CA 1.5 Specification, 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 an 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.

Using the `capacity-increment` element in the `weblogic-ra.xml` descriptor file, you can specify a number of additional `ManagedConnections` to be created automatically when a match is not found. This feature provides give you the flexibility to control connection pool growth over time and the performance hit on the server each time this growth occurs.

shrinking-enabled: 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 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 the initial capacity or as close to this as possible 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`.

shrink-frequency-seconds: Setting the Wait Time Between Attempts to Reclaim Unused ManagedConnections

Use the `shrink-frequency-seconds` element in the `weblogic-ra.xml` descriptor file to identify the amount of time (in seconds) the Connection Pool Manager will wait between attempts to reclaim unused `ManagedConnections`. The default value of this element is `900` seconds.

highest-num-waiters: Controlling the Number of Clients Waiting for a Connection

If the maximum number of connections has been reached and there are no available connections, WebLogic Server retries until the call times out. The `highest-num-waiters` element controls the number of clients that can be waiting at any given time for a connection.

highest-num-unavailable: Controlling the Number of Unavailable Connections

When a connection is created and fails, the connection is placed on an unavailable list. WebLogic Server attempts to recreate failed connections on the unavailable list. The `highest-num-unavailable` element controls the number of unavailable connections that can exist on the unavailable list at one time.

connection-creation-retry-frequency-seconds: Recreating Connections

To configure WebLogic Server to attempt to recreate a connection that fails while creating additional `ManagedConnections`, enable the `connection-creation-retry-frequency-seconds` element. By default, this feature is disabled.

match-connections-supported: Matching Connections

A connection request contains parameter information. By default, the connector container calls the `matchManagedConnections()` method on the `ManagedConnectionFactory` to match the

available connection in the pool to the parameters in the request. The connection that is successfully matched is returned.

It may be that the `ManagedConnectionFactory` does not support the call to `matchManagedConnections()`. If so, the `matchManagedConnections()` method call throws a `javax.resource.NotSupportedException`. If the exception is caught, the connector container automatically stops calling the `matchManagedConnections()` method on the `ManagedConnectionFactory`.

You can set the `match-connections-supported` element to specify whether the resource adapter supports connection matching. By default, this element is set to true and the `matchManagedConnections()` method is called at least once. If it is set to false, the method call is never made.

If connection matching is not supported, a new resource is created and returned if the maximum number of resources has not been reached; otherwise, the oldest unavailable resource is refreshed and returned.

test-frequency-seconds: Testing the Viability of Connections

The `test-frequency-seconds` element allows you to specify how frequently (in seconds) connections in the pool are tested for viability.

test-connections-on-create: Testing Connections upon Creation

You can set the `test-connections-on-create` element to enable the testing of connections as they are created. The default value is `false`.

test-connections-on-release: Testing Connections upon Release to Connection Pool

You can set the `test-connections-on-release` element to enable the testing of connections as they are released back into the connection pool. The default value is `false`.

test-connections-on-reserve: Testing Connections upon Reservation

You can set the `test-connections-on-reserve` element to enable the testing of connections as they are reserved from the connection pool. The default value is `false`.

Connection Proxy Wrapper - 1.0 Resource Adapters

The connection proxy wrapper feature is valid only for resource adapters that are created based on the J2EE 1.0 Connector Architecture. When a connection request is made, WebLogic Server returns to the client (by way of the resource adapter) a proxy object that wraps the connection object. WebLogic Server uses this proxy to provide the following features:

- Connection leak detection capabilities
- Late XAResource enlistment when a connection request is made before starting a global transaction that uses that connection

Possible ClassCastException

If the connection object returned from a connection request is cast as a `Connection` implementation class (rather than an interface implemented by the `Connection` class), a `ClassCastException` can occur. This exception is caused by one of the following:

- The resource adapter performing the cast
- The client performing the cast during a connection request

An attempt is made by WebLogic Server to detect the `ClassCastException` caused by the resource adapter. If the server detects that this cast is failing, it turns off the proxy wrapper feature and proceeds by returning the unwrapped connection object during a connection request. The server logs a warning message to indicate that proxy generation has been turned off. When this occurs, connection leak detection and late XAResource enlistment features are also turned off.

WebLogic Server attempts to detect the `ClassCastException` by performing a test at resource adapter deployment time by acting as a client using container-managed security. This requires the resource adapter to be deployed with security credentials defined.

If the client is performing the cast and receiving a `ClassCastException`, the client code can be modified, as in the following example.

Assume the client is casting the connection object to `MyConnection`.

1. Rather than having `MyConnection` be a class that implements the resource adapter's `Connection` interface, modify `MyConnection` to be an interface that extends `Connection`.
2. Implement a `MyConnectionImpl` class that implements the `MyConnection` interface.

Turning Proxy Generation On and Off

If you know for sure whether or not a connection proxy can be used in the resource adapter, you can avoid a proxy test by explicitly setting the `use-connection-proxies` element in the WebLogic Server 8.1 version of `weblogic-ra.xml` to `true` or `false`.

Note: WebLogic Server still supports J2CA 1.0 resource adapters. For 1.0 resource adapters, continue to use the WebLogic Server 8.1 deployment descriptors found in `weblogic-ra.xml`. It contains elements that continue to accommodate 1.0 resource adapters.

If set to `true`, the proxy test is not performed and connection properties are generated.

If set to `false`, the proxy test is not performed and connection proxies are generated.

If `use-connection-proxies` is unspecified, the proxy test is performed and proxies are generated if the test passes. (The test passes if a `ClassCastException` is not thrown by the resource adapter).

Note: The test cannot detect a `ClassCastException` caused by the client code.

Testing Connections

If a resource adapter's `ManagedConnectionFactory` implements the `ValidatingManagedConnectionFactory` interface, then the application server can test the validity of existing connections. You can test either a specific outbound connection or the entire pool of outbound connections for a particular `ManagedConnectionFactory`. Testing the entire pool results in testing each connection in the pool individually. For more information on this feature, see section 6.5.3.4 "Detecting Invalid Connections" in the J2CA 1.5 Specification.

Configuring Connection Testing

The following optional elements in the `weblogic-ra.xml` deployment descriptor allow you to control the testing of connections in the pool.

- `test-frequency-seconds`—The connector container periodically tests all the free connections in the pool. Use this element to specify the frequency with which the connections are tested. The default is 0, which means the connections will not be tested.

- `test-connections-on-create`—Determines whether the connection should be tested upon its creation. By default it is false.
- `test-connections-on-release`—Determines whether the connection should be tested upon its release. By default it is false.
- `test-connections-on-reserve`—Determines whether the connection should be tested upon its reservation. By default it is false.

Testing Connections in the Administration Console

To test a resource adapter's connection pools:

1. In the Administration Console, open the Deployments page and select the resource adapter in the Deployments table.
2. Select the Test tab.

   You will see a table of connection pools for the resource adapter and the test status of each pool.
3. Select the connection pool you want to test and click Test.

See Test outbound connections in the console help.

Transaction Management

The following sections discuss the system-level transaction management contract that is used for outbound communication from WebLogic Server to Enterprise Information Systems (EISes):

- "Supported Transaction Levels" on page 6-1
- "Configuring Transaction Levels" on page 6-3

For more information on transaction management, see Chapter 7 "Transaction Management" of the J2CA 1.5 Specification. For information about transaction management for inbound communication from EISes to WebLogic Server, see "Transactional Inflow" on page 7-9.

Supported Transaction Levels

A *transaction* is a set of operations that must be committed together or not at all for the data to remain consistent and to maintain data integrity. Transactional access to EISes is an important requirement for business applications. The J2EE 1.5 Connector Architecture supports the use of transactions.

WebLogic Server utilizes the WebLogic Server Transaction Manager implementation and supports resource adapters having XA, local, or no transaction support. You define the type of transaction support in the `transaction-support` element in the `ra.xml` file; a resource adapter can support only one type. You can use the `transaction-support` element in the `weblogic-ra.xml` deployment descriptor to override the value specified in `ra.xml`. See "Configuring Transaction Levels" on page 6-3 and "transaction-support" on page A-20 for details.

XA Transaction Support

XA transaction support allows a transaction to be managed by a transaction manager external to a resource adapter (and therefore external to an EIS). 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 cleans up the EIS connection once the transaction has completed.

Local Transaction Support

Local transaction support allows WebLogic Server to manage resources that are local to the resource adapter. Unlike XA transaction, local transaction generally cannot participate in a two-phase commit protocol (2PC). The only way a local transaction resource adapter can be involved in a 2PC transaction is if it is the only local transaction resource involved in the transaction and if the WebLogic Server Connector container uses a Last Resource Commit Optimization whereby the outcome of the transaction is governed by the resource adapter's local transaction.

A local transaction is normally started by using the API that is specific to that resource adapter, or the CCI interface if it is supported for that adapter. When a resource adapter connection that is configured to use local transaction support is created and used within the context of an XA transaction, WebLogic Server automatically starts a local transaction to be used for this connection. When the XA transaction completes and is ready to commit, prepare is first called on the XA resources that are part of the XA transaction. Next, the local transaction is committed.

If the commit fails on the local transaction, the XA transaction and all the XA resources are rolled back. If the commit succeeds, all the XA resources for the XA transaction are committed. When an application component closes the connection, WebLogic Server cleans up the connection once the transaction has completed.

No Transaction Support

If a resource adapter is configured to use no transaction support, the resource adapter can still be used in the context of a transaction. However, in this case, the connections used for that resource adapter are never enlisted in a transaction and behave as if no transaction was present. In other words, operations performed using these connections are made to the underlying EIS immediately, and if the transaction is rolled back, the changes are not undone for these connections.

Configuring Transaction Levels

You specify a transaction support level for a resource adapter in the J2EE standard resource adapter deployment descriptor, `ra.xml`. 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>`

The transaction support value specified in the `ra.xml` deployment descriptor is the default value for all Connection Factories of the resource adapter. You can override this value for a particular Connection Factory by specifying a value in the `transaction-support` element of the `weblogic-ra.xml` deployment descriptor.

The value of `transaction-support` must be one of the following:

- `NoTransaction`
- `LocalTransaction`
- `XATransaction`

For more information on specifying the transaction level in the `ra.xml` deployment descriptor, see Section 17.6 “Resource Adapter XML Schema Definition” of the J2CA 1.5 Specification. For more information on specifying the transaction level in the `weblogic-ra.xml` deployment descriptor, see Appendix A, “weblogic-ra.xml Schema.”

Message and Transactional Inflow

This section discusses how WebLogic resource adapters use inbound connections to handle message inflow and transactional inflow.

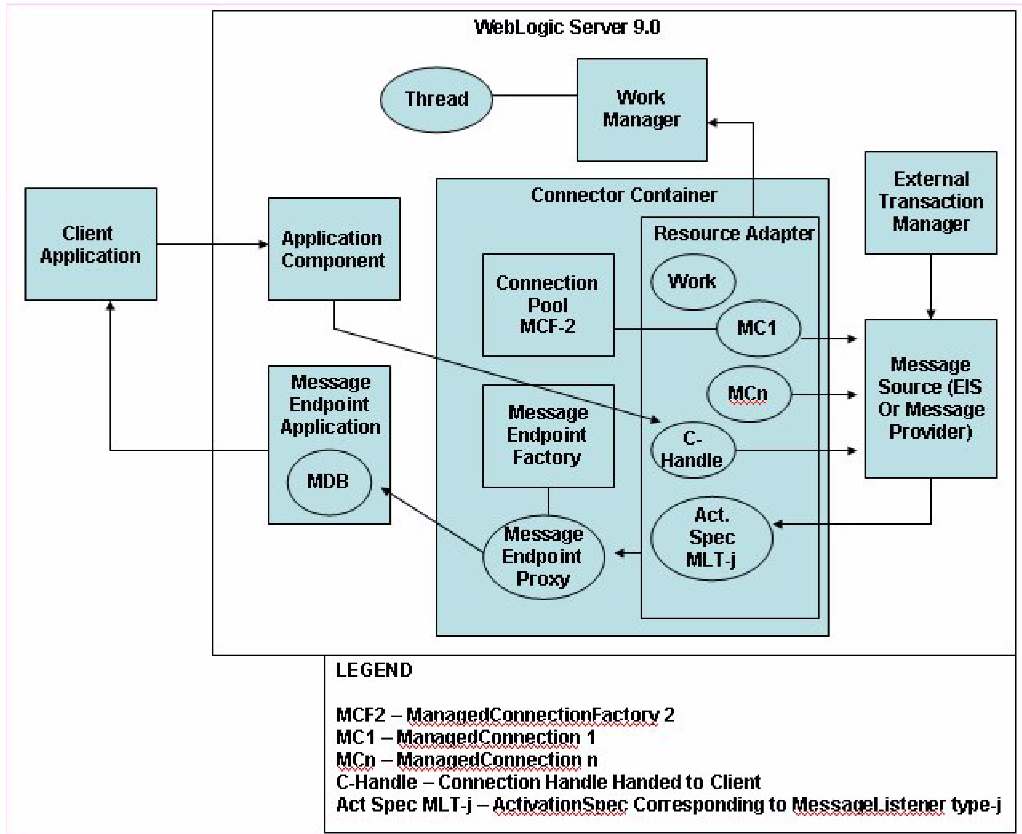
- "Overview of Message and Transactional Inflow" on page 7-1
- "How Message Inflow Works" on page 7-4
- "Message Inflow to Message Endpoints (Message-driven Beans)" on page 7-6
- "Transactional Inflow" on page 7-9

Overview of Message and Transactional Inflow

Message inflow refers to inbound communication from an EIS to the application server, using a resource adapter. Inbound messages can be part of a transaction that is governed by a Transaction Manager that is external to WebLogic Server and the resource adapter, as described in "Transactional Inflow" on page 7-9.

The following diagram provides an overview of how messaging and transaction inflow occurs within a resource adapter and the role played by the Work Manager.

Figure 7-1 Messaging and Transactional Inflow Architecture



Architecture Components

Figure 7-1 contains the following components:

- A client application, which connects to an application running on WebLogic Server, but which also needs to connect to an EIS
- An external system (in this case, an EIS or Enterprise Information System)
- An application component (an EJB) that the client application uses to submit outbound requests to the EIS through the resource adapter

- A message endpoint application (a message-driven bean and possibly other J2EE components) used for the receipt of inbound messages from the EIS through the resource adapter
- The WebLogic Server Work Manager and an associated thread (or threads) to which the resource adapter submits Work instances to process inbound messages and possibly process other actions.
- An external Transaction Manager, to which the WebLogic Server Transaction Manager is subordinate for transactional inflow of messages from the EIS
- The WebLogic Server Connector container in which the resource adapter is deployed. The container manages the following:
  - A deployed resource adapter that provides bi-directional (inbound and outbound) communication to and from the EIS.
  - An active `Work` instance.
  - Multiple managed connections (MC1, ..., MCn), which are objects representing the outbound physical connections from the resource adapter to the EIS.
  - Connection handles (C-handle) returned to the application component from the connection factory of the resource adapter and used by the application component for communicating with the EIS.
  - One of perhaps many activation specifications. There is an activation specification (`ActivationSpec`) that corresponds to each specific message listener type, MLT-j. For information about requirements for an `ActivationSpec` class, see Chapter 12, "Message Inflow" in the J2CA 1.5 Specification.
  - One of the connection pools maintained by the container for the management of managed connections for a given `ManagedConnectionFactory` (in this case, MCF-2. A Connector container could include multiple connection pools, each corresponding to a different type of connections to a single EIS or even different EISes).
  - A `MessageEndpointFactory` created by the EJB container and used by the resource adapter to create proxies to `MessageEndpoint` instances (MDB instances from the MDB pool).
- An external message source, which could be an EIS or Message Provider

Inbound Communication Scenario

This section describes a basic inbound communication scenario that may be described using the diagram, showing how inbound messages originate in an EIS, flow into the resource adapter, and are handled by a Message-driven Bean. For related information, see Figure 2-1.

A typical simplified inbound sequence involves the following steps:

1. The EIS sends a message to the resource adapter.
2. The resource adapter inspects the message and determines what type of message it is.
3. The resource adapter may create a `Work` object and submit it to the Work Manager. The Work Manager performs the succeeding work in a separate Thread, while the resource adapter can continue waiting for other incoming messages.
4. Based on the message type, the resource adapter (either directly or as part of a `Work` instance) looks up the correct message endpoint to which it will send the message.
5. Using the message endpoint factory corresponding to the type of message endpoint it needs, the resource adapter creates a message endpoint (which is a proxy to a message-driven bean instance from the MDB pool).
6. The resource adapter invokes the message listener method on the endpoint, passing it message content based on the message it received from the EIS.
7. The message is handled by the MDB in one of several possible ways:
    a. the MDB may handle the message directly and possibly return a result to the EIS through the resource adapter
    b. the MDB may distribute the message to some other application component
    c. the MDB may place the message on a queue to be picked up by the client
    d. the MDB may directly communicate with the client application.

How Message Inflow Works

A resource adapter that supports inbound communication from an EIS to the application server typically includes the following:

- A proprietary communications channel and protocol for connecting to and communicating with an EIS. The communications channel and protocol are not visible to the application

server in which the resource adapter is deployed. See "Proprietary Communications Channel and Protocol" on page 7-6.

- One or more message types recognized by the resource adapter.
- A dispatching mechanism to dispatch a message of a given type to another component in the application server.

Handling Inbound Messages

A resource adapter may handle an inbound message in a variety of ways. For example, it may:

- Handle the message locally, that is, within the `ResourceAdapter` bean, without involving other components.
- Pass the message off to another application component. For example, it may look up an EJB and invoke a method on it.
- Send the message to a message endpoint. Typically, a message endpoint is a message-driven bean (MDB). For more information, see "Message Inflow to Message Endpoints (Message-driven Beans)" on page 7-6.

Inbound messages may return a result to the EIS that is sending the message. A message requiring an immediate response is referred to as synchronous (the sending system waits for a response). This is also referred to as request-response messaging. A message that does not expect a response as part of the same exchange with the resource adapter is referred to as asynchronous or event notification-based communication. A resource adapter can support asynchronous or synchronous communications for all three destinations listed above.

Depending upon the transactional capabilities of the resource adapter and the EIS, inbound messages can be either part of a transaction (XA) or not (non-transactional). If the messages are XA, the controlling transaction may be coordinated by an external Transaction Manager (transaction *inflow*) or by the application server's Transaction Manager. See "Transactional Inflow" on page 7-9.

In most cases, inbound messages in a resource adapter are dispatched through a `Work` instance in a separate thread. The resource adapter wraps the work to be done in a `Work` instance and submits it to the application server's Work Manager for execution and management. A resource adapter can submit a Work instance using the `doWork()`, `startWork()`, or `scheduleWork()` methods depending upon the scheduling requirements of the work.

Proprietary Communications Channel and Protocol

The resource adapter can expose connection configuration information to the deployer by various means; for example, as properties on the `ResourceAdapter` bean or properties on the `ActivationSpec` object. An alternative is to use the same communication channel for inbound as well as outbound traffic. Thus you can also set configuration information on the outbound connection pool.

Message Inflow to Message Endpoints (Message-driven Beans)

Prior to EJB 2.1, a message-driven bean (MDB) supported only Java Message Service (JMS) messaging. That is, an MDB had to implement the `javax.jms.MessageListener` interface, including the `onMessage(javax.jms.Message)` message listener method. MDBs were bound to JMS components and the JMS subsystem delivered the messages to MDBs by invoking the `onMessage()` method on an instance of the MDB.

With EJB 2.1, the JMS-only MDB restriction has been lifted to accommodate the delivery of messages from inbound resource adapters. The main ingredients for message delivery to an MDB by way of a resource adapter are:

- An inbound message of a certain type (determined by the resource adapter/EIS contract)
- An `ActivationSpec` object implemented by the resource adapter
- A mapping between message types and message listener interfaces
- An MDB that implements a given message listener interface
- A deployment-time binding between an MDB and a resource adapter

For more information about message-driven Beans, see “Message-Driven EJBs” in *Programming WebLogic Enterprise JavaBeans*.

Deployment-Time Binding Between an MDB and a Resource Adapter

A resource adapter can be deployed independently (as a standalone RAR) or as part of an enterprise application (EAR). An MDB can also be deployed independently (as a standalone JAR) or as part of an enterprise application (EAR). In either case, an MDB whose messages are

derived from a resource adapter must be bound to the resource adapter. The following sections describe binding the MDB and resource adapter and subsequent messaging operations.

Binding an MDB and a Resource Adapter

To bind an MDB and a resource adapter, you must:

1. Set the `jndi-name` element in the `weblogic-ra.xml` deployment descriptor for the resource adapter. See "jndi-name" on page A-2.
2. Set the `adapter-jndi-name` element in the `weblogic-ejb-jar.xml` deployment descriptor to match the value set in the corresponding `jndi-name` element in the resource adapter.
3. Assume that the resource adapter is deployed prior to the MDB. (The MDB could be deployed before the resource adapter is deployed; in that case, the deployed MDB polls until the resource adapter is deployed.) When the resource adapter is deployed, the `ResourceAdapter` bean is bound into JNDI using the name specified.
4. The MDB is deployed, and the MDB container invokes an application server-specific API that looks up the resource adapter by its JNDI name and invokes the specification-mandated `endpointActivation(MessageEndpointFactory, ActivationSpec)` method on the resource adapter.
5. The MDB container provides the resource adapter with a configured `ActivationSpec` (containing configuration information) and a factory for the creation of message endpoint instances.
6. The resource adapter saves this information for later use in message delivery. The resource adapter thereby knows what message listener interface the MDB implements. This information is important for determining what kind of messages to deliver to the MDB.

Dispatching a Message

When a message arrives from the EIS to the resource adapter, the resource adapter determines where to dispatch it. The following is a possible sequence of events:

1. A message arrives from the EIS to the resource adapter.
2. The resource adapter examines the message and determines its type by looking it up in an internal table. The resource adapter determines the message type corresponds to a particular pair (`MessageEndpointFactory`, `ActivationSpec`).
3. The resource adapter determines the message should be dispatched to an MDB.

4. Using the `MessageEndpointFactory` for that type of message endpoint (one to be dispatched to an MDB), the resource adapter creates an MDB instance by invoking `createEndpoint()` on the factory.

5. The resource adapter then invokes the message listener method on the MDB instance, passing any required information (such as the body of the incoming message) to the MDB.

6. If the message listener does not return a value, the message dispatching process is complete.

7. If the message listener returns a value, the resource adapter determines how to handle that value. This may or may not result in further communication with the EIS, depending upon the contract with the EIS.

Activation Specifications

A resource adapter is configured with a mapping of message types and activation specifications. The activation specification is a JavaBean that implements `javax.resource.spi.ActivationSpec`. The resource adapter has an `ActivationSpec` class for each supported message type. The mapping of message types and activation specifications is configured in the `ra.xml` deployment descriptor, as described in "Configuring Inbound Connections" on page 5-9. For more information about `ActivationSpecs`, see Chapter 12, "Message Inflow" in the J2CA 1.5 Specification.

Administered Objects

As described in section 12.4.2.3 of the J2CA 1.5 Specification, a resource adapter may provide the Java class name and the interface type of an optional set of JavaBean classes representing administered objects that are specific to a messaging style or message provider. You configure administered objects in the `admin-objects` elements of the `ra.xml` and `weblogic-ra.xml` deployment descriptor files. As with outbound connections and other WebLogic resource adapter configuration elements, you can define administered objects at three configuration scope levels:

- Global—Specify parameters that apply to all administered objects in the resource adapter using the `default-properties` element. See "default-properties" on page A-16.

- Group—Specify parameters that apply to all administered objects belonging to a particular administered object group specified in the `ra.xml` deployment descriptor using the `admin-object-group` element. The properties specified in a group override any parameters specified at the global level. See "admin-object-group" on page A-15.

  The `admin-object-interface` element (a subelement of the `admin-object-group` element) serves as a required unique element (a key) to each `admin-object-group`.

There must be a one-to-one relationship between the `admin-object-interface` element in `weblogic-ra.xml` and the `admin-object-interface` element in `ra.xml`.

- Instance—Under each admin object group, you can specify administered object instances using the `admin-object-instance` element of the `weblogic-ra.xml` deployment descriptor. These correspond to the individual administered objects for the resource adapter. You can use the `admin-object-properties` subelement to specify properties at the instance level too; properties specified at the instance level override those provided at the group and global levels. See "admin-object-instance" on page A-17.

Transactional Inflow

This section discusses how transactions flow into WebLogic Server from an EIS and a resource adapter. A transaction inflow contract allows the resource adapter to handle transaction completion and crash recovery calls initiated by an EIS. It also ensures that ACID properties of the imported transaction are preserved. For more information on transaction inflow, see Chapter 14, "Transaction Inflow" of the J2CA 1.5 Specification.

When an EIS passes a message through a resource adapter to the application server, it may pass a transactional context under which messages are delivered or work is performed. The inbound transaction will be controlled by a transaction manager external to the resource adapter and application server. See "Message Inflow to Message Endpoints (Message-driven Beans)" on page 7-6.

A resource adapter may act as a bridge between the EIS and the application server for transactional control. That is, the resource adapter receives messages that it interprets as XA callbacks for participating in a transaction with a external Transaction Manager.

WebLogic Server can function as an XA resource to a external Transaction Manager through its interposed Transaction Manager. The WebLogic Server Transaction Manager maps external transaction IDs to WebLogic Server-specific transaction IDs for such transactions.

The WebLogic Server Transaction Manager is subordinate to the external Transaction Manager, which means that the external Transaction Manager ultimately determines whether the transaction succeeds or is rolled back. See "Participating in Transactions Managed by a Third-Party Transaction Manager" in *Programming WebLogic JTA*. As part of the J2EE 1.5 Connector Architecture, the ability for a resource adapter to participate in such a transaction is now exposed through a J2EE standard API.

The following illustrates how a resource adapter would participate in a external transaction. For more information, see section 14.4, "Transaction Inflow Model" of the J2CA 1.5 Specification.

1. The resource adapter receives an inbound message with a new external transaction ID.
2. The resource adapter decodes the external transaction ID and constructs an `Xid` (`javax.transaction.xa.Xid`).
3. The resource adapter creates an instance of an `ExecutionContext` (`javax.resource.spi.work.ExecutionContext`), setting the `Xid` it created and also setting a transaction timeout value.
4. The resource adapter creates a new `Work` object to process the incoming message and deliver it to a message endpoint.
5. The resource adapter submits the `Work` object and the `ExecutionContext` to the Work Manager for processing. At this point, the Work Manager performs the necessary work to enlist the transaction and start it with the WebLogic Server Transaction Manager.
6. Subsequent XA calls from the external Transaction Manager are sent through the resource adapter and communicated to the WebLogic Server Transaction Manager. In this way, the resource adapter acts as a bridge for the XA calls between the external Transaction Manager and the WebLogic Server Transaction Manager, which is acting as a resource manager.

Using the Transactional Inflow Model for Locally Managed Transactions

When the resource adapter receives requests from application components running in the same server instance as the resource adapter that need to be delivered to an MDB as part of the same transaction as the resource adapter request, the transaction ID must be obtained from the transaction on the current thread and placed in an `ExecutionContext`.

In this case, WebLogic Server does not use the Interposed Transaction Manager but simply passes the transaction on to the Work Thread used for message delivery to the MDB.

Security

Since a resource adapter needs to be able to establish connections with external systems, it needs to be configured with authentication and other security information necessary to make the connections. The following sections discuss WebLogic Server resource adapter security for outbound communication:

- "Container-Managed and Application-Managed Sign-on" on page 8-1
- "Password Credential Mapping" on page 8-2
- "Security Policy Processing" on page 8-5
- "Configuring Security Identities for Resource Adapters" on page 8-6
- "Configuring Connection Factory-Specific Authentication and Re-authentication Mechanisms" on page 8-11

For more information about WebLogic security, see *Understanding WebLogic Security* and *Securing WebLogic Resources*.

Container-Managed and Application-Managed Sign-on

When a resource adapter makes an outbound connection to an Enterprise Information System (EIS), it needs to sign on with valid security credentials. In accordance with the J2CA 1.5 Specification, WebLogic Server supports both container-managed and application-managed sign-on for outbound connections. At runtime, WebLogic Server determines the chosen sign-on mechanism, based on the information specified in either the invoking client component's deployment descriptor or the `res-auth` element of the resource adapter deployment descriptor.

A sign-on mechanism specified in a resource adapter's deployment descriptor takes precedence over one specified in the calling component's deployment descriptor. Even when using container-managed sign-on, any security information explicitly specified by the client component is presented on the call to obtain the connection.

If the Weblogic Server J2EE 1.5 Connector Architecture implementation cannot determine which sign-on mechanism is being requested by the client component, the connector container attempts container-managed sign-on.

Application-Managed Sign-on

With application-managed sign-on, the client component supplies the necessary security credentials (typically a user name and password) when making the call to obtain a connection to an EIS. In this scenario, the application server provides no additional security processing other than to pass along this information in the request for the connection.

Container-Managed Sign-on

WebLogic Server and an EIS each maintain independent security realms. A goal of container-managed sign-on is to permit a user to sign on to WebLogic Server and be able to use applications that access an EIS through a resource adapter without having to sign on separately to the EIS. Container-managed sign-on in WebLogic Server uses *credential mappings*, which map credentials (either username/password pairs or security tokens) of WebLogic security principals (which may be either authenticated individual users or client applications) to the corresponding credentials required to access the EIS. For any deployed resource adapter, you can configure credential mappings for applicable security principals. For related information, see "Credential Mappings" on page 8-3.

Password Credential Mapping

The J2CA 1.5 Specification requires storage of credentials in a `javax.security.auth.Subject`. The credentials are passed to either the `createManagedConnection()` or the `matchManagedConnection()` methods of the `ManagedConnectionFactory` object. Credential mapping information is stored in the WebLogic Server embedded LDAP storage. Credential mappings are specific to outbound resource adapters.

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.

Password authentication is the only authentication mechanism supported by WebLogic Server out of the box. Password authentication consists of a user ID and password. Based on the configured mappings, when a user requests connection to a resource adapter, the appropriate credentials for that user are supplied to the resource adapter.

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 clear text. Using SSL, WebLogic Server can authenticate the user without compromising the confidentiality of the user's ID and password. For more information, see "Configuring SSL" in *Securing WebLogic Server*.

Credential Mappings

Credential mappings are specific to outbound resource adapters. You configure credential mappings using the WebLogic Server Administration Console. Before you can configure credential mappings, however, you must successfully deploy the resource adapter. Note that the first time you deploy a resource adapter, it has no credential mappings configured and initial connections will fail until these are configured.

If the resource adapter requires credentials and is configured to create connections at deployment time (meaning the `initial-capacity` element in the `weblogic-ra.xml` is set to greater than `0`), this may cause the initial connection to fail. Therefore, BEA recommends that—for the initial installation and deployment of this resource adapter—you set the `initial-capacity` element to `0` for its connection pool. After you configure the appropriate credentials and after the initial deployment of the resource adapter, you can change the value of the `initial-capacity` element. For more information, see "initial-capacity: Setting the Initial Number of ManagedConnections" on page 5-11.

You can configure credential mappings for individual outbound connection pools or globally for all the connection pools in the resource adapter. When the resource adapter receives a request for a connection, WebLogic Server searches for credential mappings configured for a specific connection pool and then checks the mappings configured globally for the resource adapter. The server searches for mappings in the following order:

1. Specific mappings at the connection factory level.
2. Specific mappings at the global level.
3. Default mappings at the connection factory level.
4. Default mappings at the global level.

For example, consider two connection pools with the following credential mappings:

Listing 8-1 Credential Mapping Examples

```
poolA
       system user name: admin
       system password: adminpw
       default user name: guest1
       default password: guest1pw1

poolB
       wlsjoe user name: harry
       wlsjoe password: harrypw

global
       system user name: sysman
       system password: sysmanpw
       wlsjoe user name: scott
       wlsjoe password: tiger
       default user name: viewer
       default password: viewerpw
```

Referring to the example provided in Listing 8-1, consider an application authenticated as `system` that makes a connection request against `poolA`. Because a specific credential mapping is defined for `system` for `poolA`, the resource adapter uses this mapping (`admin/adminpw`).

If the application makes the same request against `poolB` as `system`, there is no corresponding specific credential mapping for `system`; therefore, the server searches for the credential mapping at the `global` level where it finds a mapping (`sysman/sysmanpw`).

If another application authenticates as `wlsjoe` and makes a request against `poolA`, it finds no mapping for `wlsjoe` defined for `poolA`. It then searches at the global level and finds a mapping for `wlsjoe` (`scott/tiger`). Against `poolB`, the application would find the mapping defined for `poolB` (`harry/harrypw`).

If an application authenticated as `user1` makes a request against `poolA`, it finds no mapping for `user1` for `poolA`. The following sequence occurs:

1. The application searches at the global level, which also has no mapping for `user1`.
2. The application searches the `poolA` mappings for a default mapping and finds a default mapping.

Creating Credential Mappings Using the Console

You can create credential maps with the WebLogic Server Administration Console. If you are using the WebLogic Credential Mapping provider, the credential maps are stored in the embedded LDAP server. For information about how to create a credential map, see Create credential mappings in the Administration Console online help.

Security Policy Processing

A *security policy* is an association between a WebLogic resource and one or more users, groups, or security roles and is designed to protect the WebLogic resource against unauthorized access. The J2CA 1.5 Specification defines default security policies for resource adapters running in an application server. It also defines how resource adapters can provide their own specific security policies overriding the default. The `weblogic.policy` file that ships with WebLogic Server establishes the default security policies as specified in the J2CA 1.5 Specification.

If the resource adapter does not have a specific security policy defined, WebLogic Server establishes the runtime environment for the resource adapter with the default security policies specified in the `weblogic.policy` file, which conforms to the defaults specified by the J2CA 1.5 Specification. If the resource adapter has defined specific security policies, WebLogic Server

establishes the runtime environment for the resource adapter with a combination of the default security policies for resource adapters and the specific policies defined for the resource adapter. You define specific security policies for resource adapters 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 Chapter 18, "Runtime Environment" in the J2CA 1.5 Specification. For more information about security policies and the WebLogic security framework, see " Security Policies" in *Securing WebLogic Resources.*

Configuring Security Identities for Resource Adapters

This section describes how to configure various security identities for WebLogic Server resource adapters in the `weblogic-ra.xml` deployment descriptor. Security identities determine which security principals can perform particular resource adapter functions. In a WebLogic resource adapter, you can either have a single security identity that can perform all functions, or use separate identities for separate classes of functions. You can define the following four types of security identities in the `weblogic-ra.xml` deployment descriptor:

- default principal—a security principal that can perform all resource adapter tasks.
- run-as principal—a security principal used by calls from the connector container into the resource adapter code during connection requests.
- run-work-as principal—a security principal used for Work instances launched by the resource adapter.
- manage-as principal—a security principal used for resource adapter management tasks, such as startup, shutdown, testing, and transaction management.

Listing 8-2 is an excerpt from a `weblogic-ra.xml` deployment descriptor that illustrates how you would configure all four of these available security identities for performing different resource adapter tasks.

Listing 8-2 Configuring All Security Identities for Resource Adapters

```
<weblogic-connector xmlns="http://www.bea.com/ns/weblogic/90">
   <jndi-name>900blackbox-notx</jndi-name>
   <security>
```

```
      <default-principal-name>
         <principal-name>system</principal-name>
      </default-principal-name>
      <run-as-principal-name>
         <principal-name>raruser</principal-name>
      </run-as-principal-name>
      <run-work-as-principal-name>
         <principal-name>workuser</principal-name>
      </run-work-as-principal-name>
      <manage-as-principal-name>
         <principal-name>raruser</principal-name>
      </manage-as-principal-name>
   </security>
</weblogic-connector>
```

Listing 8-3 illustrates how you could use the `<default-principal-name>` element to configure a single default principal security identity for performing all resource adapter tasks.

Listing 8-3 Configuring a Single Default Principal Identity for a Resource Adapter

```
<weblogic-connector xmlns="http://www.bea.com/ns/weblogic/90">
   <jndi-name>900blackbox-notx</jndi-name>
   <security>
      <default-principal-name>
         <principal-name>system</principal-name>
      </default-principal-name>
   </security>
</weblogic-connector>
```

For more information on setting security identity properties, see "security" on page A-10.

default-principal-name: Default Identity

You can define a single security identity that can be used for all resource adapter purposes using the `default-principal-name` element. If values are not specified for `run-as-principal-name`, `manage-as-principal-name`, and `run-work-as-principal-name`, they default to the value set for `default-principal-name`.

The value of `default-principal-name` can be set to a defined WebLogic Server user name such as `system` or to use an anonymous identity (which is the equivalent of having no security identity).

For example, you can create a single security identity that makes all calls from WebLogic Server into the resource adapter and manages all resource adapter management tasks with a default `system` identity as follows:

Listing 8-4 Using a Defined WebLogic Server Name

```
<security>
   <default-principal-name>
      <principal-name>system</principal-name>
   </default-principal-name>
</security>
```

You can set the `default-principal-name` element to `anonymous` as follows:

Listing 8-5 Setting Up an Anonymous Identity

```
<security>
   <default-principal-name>
      <use-anonymous-identity>true</use-anonymous-identity>
```

```
   </default-principal-name>
</security>
```

manage-as-principal-name: Identity for Running Management Tasks

You can define a management identity that is used for running various resource adapter management tasks such as startup, shutdown, testing, shrinking, and transaction management using the `manage-as-principal-name` element.

As with `default-principal-name`, the value of `manage-as-principal-name` can be set to a defined WebLogic Server user name such as `system` or to use an `anonymous` identity (which is the equivalent of having no security identity). If you do not set up a value for the `manage-as-principal-name` element, it defaults to the value set up for `default-principal-name`. If no value is set up for `default-principal-name`, it defaults to the `anonymous` identity.

Listing 8-6 illustrates how you can configure a resource adapter to run management calls using the WebLogic Server-defined user name `system`.

Listing 8-6 Using a Defined WebLogic Server Name

```
<security>
   <manage-as-principal-name>
      <principal-name>system</principal-name>
   </manage-as-principal-name>
</security>
```

Listing 8-7 illustrates how you can configure a resource adapter to run management calls using an `anonymous` identity.

Listing 8-7 Setting Up an Anonymous Identity

```
<security>
   <manage-as-principal-name>
      <use-anonymous-identity>true</use-anonymous-identity>
   </manage-as-principal-name>
</security>
```

run-as-principal-name: Identity Used for Connection Calls from the Connector Container into the Resource Adapter

You define the principal name that should be used by all calls from the connector container into the resource adapter code during connection requests in the `run-as-principal-name` element. This principal name is used, for example, when resource adapter objects such as the `ManagedConnectionFactory` are instantiated—in other words, when the WebLogic Server connector container makes calls to the resource adapter, the identity defined in the `run-as-principal-name` element is used. This may include calls as part of either inbound or outbound requests or setup, or as part of initialization not specific to either inbound or outbound resource adapters (for example, `ResourceAdapter.start()`).

The value of the `run-as-principal-name` element can be set in one of three ways:

- To a defined WebLogic Server name
- To use an anonymous identity
- To use the security identity of the calling code.

If the value of the `run-as-principal-name` element is not defined, it defaults to the value of the `default-principal-name` element. If the `default-principal-name` element is not defined, it defaults to the identity of the requesting caller.

run-work-as-principal-name: Identity Used for Performing Resource Adapter Management Tasks

For inbound resource adapters, BEA recommends that you use Work instances to execute inbound requests. To establish the security identity for Work instances launched by a resource

adapter, you specify this value using the `run-work-as-principal-name` element. However, Work instances can also be created as part of outbound resource adapters to execute outbound requests. If an adapter does not use Work instances to handle inbound requests, then inbound requests are either run with no security context established (`anonymous`) or the resource adapter can make WebLogic Server-specific calls to authenticate as a WebLogic Server user. In this case, the WebLogic Server user security context is used.

The value of the `run-work-as-principal-name` element can be set in one of three ways:

- To a defined WebLogic Server name
- To use an anonymous identity
- To use the security identity of the calling code

If the value of the `run-work-as-principal-name` element is not defined, it defaults to the value of the `default-principal-name` element. If the `default-principal-name` element is not defined, it defaults to the identity of the requesting caller.

To run work using the requesting caller's identity, you specify the `run-work-as-principal-name` element as follows:

Listing 8-8 Using the Requesting Caller's Identity

```
<security>
   <run-work-as-principal-name>
      <use-caller-identity>true</use-caller-identity>
   </run-work-as-principal-name>
</security>
```

Configuring Connection Factory-Specific Authentication and Re-authentication Mechanisms

You specify authentication and re-authentication mechanisms for a resource adapter in the J2EE standard resource adapter deployment descriptor, `ra.xml`. These settings apply to all outbound connection factories.

- The `authentication-mechanism` element specifies an authentication mechanism to be used by all outbound connection factories.
- The `reauthentication-support` element specifies whether outbound connection factories support re-authentication of existing Managed-Connection instances. This is intended to be the default value for all connection factories of the resource adapter.

You can override the `authentication-mechanism` and `reauthentication-support` values in the `ra.xml` deployment descriptor by specifying them in the `weblogic-ra.xml` deployment descriptor. Doing so allows you to apply these settings to a specific connection factory rather than to all connection factories. See "authentication-mechanism" on page A-20 and "reauthentication-support" on page A-21.

CHAPTER 9

Packaging and Deploying Resource Adapters

The following sections describe how to package and deploy resource adapters:

- "Packaging Resource Adapters" on page 9-1
- "Deploying Resource Adapters" on page 9-4

Deploying applications on WebLogic Server is covered in more detail in Deploying and Packaging from a Split Development Directory in *Developing Applications with WebLogic Server*.

Packaging Resource Adapters

For production and development purposes, BEA recommends packaging your assembled resource adapter (RAR) as part of an enterprise application (EAR).

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 the same directory format, whether the resource adapter is packaged in an exploded directory format or as a RAR. A typical directory structure of a resource adapter is shown in Listing 9-1:

Listing 9-1 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 directory 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.
- A resource adapter deployed in WebLogic Server supports the `class-path` entry in `MANIFEST.MF` to reference a class or resource such as a property.
- 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`.) Ensure that any dependencies of a resource adapter on platform-specific native libraries are resolved.

- 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`.)
- 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. This specification-compliant behavior may be overridden if required.

Packaging Limitation

If you reload a standalone resource adapter without reloading the client that is using it, the client may cease to function properly. This limitation is due to the J2CA 1.5 Specification limitation of not providing a remotable interface.

Packaging Resource Adapter Archives (RARs)

After you stage one or more resource adapters in a directory, you package them in a Java Archive (JAR) with a `.rar` file extension.

Note: Once you have assembled the resource adapter, BEA recommends that you package it as part of an enterprise application. This allows you to take advantage of the split development directory structure, which provides a number of benefits over the traditional single directory structure. See Creating a Split Development Directory Environment in *Developing Applications with WebLogic Server*.

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/xml/ns/j2ee/connector_1_5.xsd

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 Schema," for information on the contents of the `weblogic-ra.xml` file.

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 Configuring Resource Adapters: Main Steps" on page 3-1.

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 stand-alone resource adapter (or a resource adapter packaged as part of an enterprise application) using any one of these tools:

- WebLogic Server Administration Console
- `weblogic.Deployer` tool
- `wldeploy` Ant task
- WebLogic Scripting Tool (WLST)

For information about these application deployment techniques, see Deploying Applications and Modules in *Deploying Applications to WebLogic Server.*

You can use a deployment plan to deploy a resource adapter deployment. For a resource adapter, a WebLogic Server deployment plan is an optional XML document that resides outside of the RAR and configures the resource adapter for deployment to a specific WebLogic Server environment. A deployment plan works by setting deployment property values that would normally be defined in the resource adapter's deployment descriptors, or by overriding property values that are already defined in the deployment descriptors. For information on deployment plans, see Configuring Applications for Production Deployment in *Deploying Applications to WebLogic Server*.

You can also deploy a resource adapter using auto-deployment. This may be useful during development and early testing. For more information, see Auto-Deploying Applications in Development Domains in *Deploying Applications to WebLogic Server.*

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.

Production Redeployment

Using WebLogic Server's production redeployment feature, you can redeploy a new version of a WebLogic Server application alongside an older version of the same application. By default, WebLogic Server immediately routes new client requests to the new version of the application, while routing existing client connections to the older version. After all clients using the older application version complete their work, WebLogic Server retires the older application so that only the new application version is active.

Suspendable Interface and Production Redeployment

Typically, a resource adapter bean implements the `javax.resource.spi.ResourceAdapter` interface. This interface defines `start()` and `stop()` methods. This type of resource adapter is not eligible for production redeployment. Resource adapters connect to one or more EISes for incoming/outgoing communication. All communication is performed in a resource adapter-proprietary way with no knowledge of the application server. If on-the-fly production redeployment is attempted, the application server can only provide notifications to the resource adapter to manage the migration of connections from the existing resource adapter to a new instance. However, the resource adapter can implement the `Suspendable` interface, which provides the capability to allow resource adapters to participate in production redeployment. For information about implementing the `Suspendable` interface, see "Suspending and Resuming Resource Adapter Activity" on page 4-7.

Production Redeployment Requirements

All of the following requirements must be met by both the old and new version of the resource adapter in order for production redeployment to work; otherwise, the redeployment fails.

- The resource adapter must be based on the J2CA 1.5 Specification. (Support for production redeployment of 1.0 resource adapters is not available.)
- The resource adapter must implement the `Suspendable` interface (see Listing 4-3).
- The resource adapter must be packaged inside an enterprise application (EAR file). Production redeployment of standalone resource adapters is not supported.
- The `Suspendable.supportsVersioning()` method must return `true` when invoked by WebLogic Server.
- The `enable-access-outside-app` element in the `weblogic-ra.xml` descriptor must be set to `false`.

Production Redeployment Process

The following process assumes the older version of the resource adapter is deployed and running. It also assumes that the older version (named `old`) as well as the newer version (named `new`) of the resource adapter meet all of the requirements mentioned in "Production Redeployment Requirements" on page 9-6 as well as the application requirements described in Updating Applications in a Production Environment in *Deploying Applications to WebLogic Server*.

The following calls are made into the resource adapters during production redeployment:

1. WebLogic Server calls `new.init(old, null)` to inform the new resource adapter that it is replacing the old resource adapter.
2. WebLogic Server calls `old.startVersioning(new, null)` to inform the old resource adapter to start its production redeployment operation with the new resource adapter.
3. WebLogic Server calls `new.start(extendedBootstrapContext)`. See "Extended BootstrapContext" on page 4-13.
4. When the old resource adapter is "finished" (meaning it has succeeded in migrating all clients and inbound connections to the new resource adapter), it calls `(ExtendedBootstrapContext)bsCtx.complete()`. This informs WebLogic Server that it is safe to undeploy the old resource adapter.
5. When undeployment occurs, WebLogic Server calls `old.stop()` and production redeployment is complete.

The calls to `new.init()` and `old.startVersioning()` give the old and new resource adapters an opportunity to migrate inbound or outbound communications from the old to the new resource adapter. How this is done is up to the individual resource adapter developer.

weblogic-ra.xml Schema

The following sections in this appendix describe the deployment descriptor elements that can be defined in the WebLogic Server-specific deployment descriptor `weblogic-ra.xml`. The schema for `weblogic-ra.xml` is http://www.bea.com/ns/weblogic/90/weblogic-ra.xsd. If your resource adapter archive (RAR) does not contain a `weblogic-ra.xml` deployment descriptor, WebLogic Server automatically selects the default values of the deployment descriptor elements.

- "weblogic-connector" on page A-2
- "work-manager" on page A-6
- "security" on page A-10
- "properties" on page A-13
- "admin-objects" on page A-14
- "outbound-resource-adapter" on page A-18

weblogic-connector

The `weblogic-connector` element is the root element of the WebLogic-specific deployment descriptor for the deployed resource adapter. You can define the following elements within the `weblogic-connector` element.

Table 9-1 weblogic-connector sublements

Element	Required/ Optional	Description
`native-libdir`	Required if native libraries are present.	Specifies the directory where all the native libraries exist that are required by the resource adapter.
`jndi-name`	Required only if a resource adapter bean is specified.	Specifies the JNDI name for the resource adapter. The resource adapter bean is registered into the JNDI tree with this name. It is not a required element if no resource adapter bean is specified. It is not a functional element if a JNDI name is specified for a resource adapter without a resource adapter bean.
`enable-access-outside-app`	Optional	As stated by the J2CA 1.5 Specification, if the resource adapter is packaged within an application (in other words, within an EAR), only components within the application should have access to the resource adapter. This element allows you to override this functionality. **Note:** This element does not apply for stand-alone resource adapters. Default Value: `false` When set to false, the resource adapter can *only* be accessed by clients that reside within the *same* application in which the resource adapter resides. **Note:** For version 1.0 resource adapters (supported in this release), the default value for this element is set to `true`.

Table 9-1 weblogic-connector sublements

Element	Required/ Optional	Description
enable-global-access-to-classes	Optional	When set to true (default), the resource adapter allows global access to its classes.
work-manager	Optional	This complex element is used to specify all the configurable elements for creating the Work Manager that will be used by the resource adapter bean. The work-manager element is imported from the weblogic-j2ee.xsd schema. The Work Manager dynamically adjusts the number of work threads to avoid deadlocks and achieve optimal throughput subject to concurrency constraints. It also meets objectives for response time goals, shares, and priorities. For subelements of work-manager, see "work-manager" on page A-6.
security	Optional	This complex element is used to specify all the security parameters for the operation of the resource adapter. See "security" on page A-10 for information on the security defaults that will be taken by the connector container.
properties	Optional	This complex element is used to override any properties that have been specified for the resource adapter bean in the ra.xml file. For subelements of properties, see "properties" on page A-13.

Table 9-1 weblogic-connector sublements

Element	Required/ Optional	Description
`admin-objects`	Optional	This complex element defines all of the admin objects in a resource adapter. As with the `outbound-resource-adapter` complex element, the `admin-objects` complex element has three hierarchical property levels that specify the configuration scope: 1. Global level—at this level, you specify parameters that apply to all admin objects in the resource adapter; you do so using the `default-properties` element. See "default-properties" on page A-15. 2. Group level—at this level, you specify parameters that apply to all admin objects belonging to a particular admin object group specified in the `ra.xml` deployment descriptor; you do so using the `admin-object-group` element. The properties specified in the group override any parameters that are specified at the global level. See "admin-object-group" on page A-15. 3. Instance level—Under each admin object group, you can use the `admin-object-instance` element to specify admin object instances. These correspond to the admin object instances for the resource adapter. You can specify properties at the instance level and override those properties provided in the group and global levels. See "admin-object-instance" on page A-16. For `admin-objects` subelements, see "admin-objects" on page A-14.

Table 9-1 weblogic-connector sublements

Element	Required/ Optional	Description
outbound-resource-adapter	Optional	This complex element is used to describe the outbound components of a resource adapter. As with the admin-objects complex element, this element has three hierarchical property levels that specify the configuration scope for defining outbound connection pools: 1. Global level—at this level, you specify parameters that apply to all outbound connection pools in the resource adapter using the default-connection-properties element. See “default-connection-properties” on page A-19. 2. Group level—at this level, you specify parameters that apply to all outbound connections belonging to a particular connection factory specified in the ra.xml deployment descriptor using the connection-definition-group element. A one-to-one correspondence exists from a connection factory in ra.xml to a connection definition group in weblogic-ra.xml. The properties specified in a group override any parameters specified at the global level. See “connection-definition-group” on page A-28. 3. The instance level—Under each connection definition group, you can specify connection instances. These correspond to the individual connection pools for the resource adapter. Parameters can be specified at this level too and these override those provided at the group and global levels. See “connection-instance” on page A-29. For outbound-resource-adapter subelements, see “outbound-resource-adapter” on page A-18.

work-manager

The `work-manager` element is a complex element that is used to specify all the configurable elements for creating the Work Manager that will be used by the resource adapter bean. The `work-manager` element is imported from the `weblogic-j2ee.xsd` schema. The following subelements can be configured in the `work-manager` element.

Table 9-2 work-manager sublements

Element	Required Optional	Description
`name`	Required	Specifies the name of the Work Manager. The J2CA 1.5 Specification describes how a resource adapter can submit work threads to the application server. These work threads are managed by the WebLogic Server Work Manager. The Work Manager dynamically adjusts the number of work threads to avoid deadlocks and achieve optimal throughput subject to concurrency constraints. It also meets objectives for response time goals, shares, and priorities.

Table 9-2 work-manager sublements

Element	Required Optional	Description
response-time-request-class fair-share-request-class context-request-class request-class-name	Optional	A work-manager element can include one and only one of the following four elements: response-time-request-class—Defines the response time request class for the application. Response time is defined with attribute goal-ms in milliseconds. The increment is ((goal - T) Cr)/R, where T is the average thread use time, R the arrival rate, and Cr a coefficient to prioritize response time goals over fair shares. fair-share-request-class—Defines the fair share request class. Fair share is defined with attribute percentage of default share. Therefore, the default is 100. The increment is Cf/(P R T), where P is the percentage, R the arrival rate, T the average thread use time, and Cf a coefficient for fair shares to prioritize them lower than response time goals. context-request-class—Defines the context class. Context is defined with multiple cases mapping contextual information, like current user or its role, cookie, or work area fields to named service classes. request-class-name—Defines the request class name.

Table 9-2 work-manager sublements

Element	Required Optional	Description
`min-threads-constraint` `min-threads-constraint-name`	Optional	You can choose between the following two elements: `min-threads-constraint`—Used to guarantee a number of threads the server allocates to requests of the constrained work set to avoid deadlocks. The default is zero. A min-threads value of one is useful, for example, for a replication update request, which is called synchronously from a peer. `min-threads-constraint-name`—Defines a name for the `min-threads-constraint` element.
`max-threads-constraint` `max-threads-constraint-name`	Optional	You can choose between the following two elements: `max-threads-constraint`—Limits the number of concurrent threads executing requests from the constrained work set. The default is unlimited. For example, consider a constraint defined with maximum threads of 10 and shared by 3 entry points. The scheduling logic ensures that not more than 10 threads are executing requests from the three entry points combined. `max-threads-constraint-name`—Defines a name for the `max-threads-constraint` element.

Table 9-2 work-manager sublements

Element	Required Optional	Description
capacity capacity-name	Optional	You can choose between the following two elements: capacity—Constraints can be defined and applied to sets of entry points, called constrained work sets. The server starts rejecting requests only when the capacity is reached. The default is zero. Note that the capacity includes all requests, queued or executing, from the constrained work set. This constraint is primarily intended for subsystems like JMS, which do their own flow control. This constraint is independent of the global queue threshold. capacity-name—Defines a name for the capacity element.

security

The `security` complex element contains default security information that can be configured for the connector container. For more information, see "Configuring Security Identities for Resource Adapters" on page 8-6.

Table 9-3 security sublements

Element	Required Optional	Description
`default-principal-name`	Optional	Specifies the default secure ID to be used for calls into the resource adapter. If this value is not specified, the default is the `anonymous` identity, which is the same as no security identity. See "default-principal-name" on page A-12 for subelements of this element.
`manage-as-principal-name`	Optional	Specifies the secure ID to be used for running various resource adapter management tasks, including startup, shutdown, testing, shrinking, and transaction management. If not specified, it defaults to the `default-principal-name` value. If `default-principal-name` is not specified, it defaults to the `anonymous` identity. See "manage-as-principal-name" on page A-12 for subelements of this element.

Table 9-3 security sublements

Element	Required Optional	Description
`run-as-principal-name`	Optional	Specifies the secure ID to be used by all calls from the connector container into the resource adapter code during connection requests. (This element currently applies only to outbound functions.) If not specified, it defaults to the `default-principal-name` value. If `default-principal-name` is not specified, it uses the identity of the requesting caller. See "run-as-principal-name" on page A-13 for subelements of this element.
`run-work-as-principal-name`	Optional	Specifies the secure ID to be used to run all work instances started by the resource adapter. If not specified, it defaults to the `default-principal-name` value. If `default-principal-name` is not specified, it uses the identity that was used to start the work. See "run-work-as-principal-name" on page A-13 for subelements of this element.

default-principal-name

The `default-principal-name` element contains the following subelements.

Table 9-4 default-principal-name sublements

Element	Required Optional	Description
`use-anonymous-identity`	Required	Specifies that the anonymous identity should be used.
`principal-name`	Required	Specifies that the principal name should be used. This should match a defined WebLogic Server user name.

manage-as-principal-name

The `manage-as-principal-name` element contains the following subelements.

Table 9-5 manage-as-principal-name sublements

Element	Required Optional	Description
`use-anonymous-identity`	Required	Specifies that the anonymous identity should be used.
`principal-name`	Required	Specifies that the principal name should be used. This should match a defined WebLogic Server user name.

run-as-principal-name

The `run-as-principal-name` element contains the following subelements.

Table 9-6 run-as-principal-name sublements

Element	Required Optional	Description
`use-anonymous-identity`	Required	Specifies that the anonymous identity should be used.
`principal-name`	Required	Specifies that the principal name should be used. This should match a defined WebLogic Server user name.
`use-caller-identity`	Required	Specifies that the caller's identity should be used.

run-work-as-principal-name

The `run-work-as-principal-name` element contains the following subelements.

Table 9-7 run-work-as-principal-name sublements

Element	Required Optional	Description
`use-anonymous-identity`	Required	Specifies that the `anonymous` identity should be used.
`principal-name`	Required	Specifies that the principal name should be used. This should match a defined WebLogic Server user name.
`use-caller-identity`	Required	Specifies that the caller's identity should be used.

properties

The `properties` element, a subelement of `weblogic-connector`, is a container for properties specified for the resource adapter bean in `ra.xml`. It holds one more or more `property` elements.

You define property elements within the properties element as follows.

Table 9-8 properties subelements

Element	Required Optional	Description
property	Required	The property element is used to override a property that has been specified for the resource adapter bean in the ra.xml file. It holds two subelements: name—Specifies the same name as the config-property-name element (a subelement of config-property in the ra.xml deployment descriptor). Setting this parameter causes the associated config-property-value element in ra.xml to be overridden. This is a required element. value—Specifies the value that overrides config-property-value element (a subelement of config-property in the ra.xml deployment descriptor). This is an optional element.

admin-objects

The admin-objects complex element defines all of the admin objects in the resource adapter. As with the outbound-resource-adapter complex element, the admin-objects complex element has three hierarchical property levels that you can specify.

The admin-objects element is a sub-element of the weblogic-connector element. You can define the following elements within the admin-objects element.

Table 9-9 admin-objects subelements

Element	Required Optional	Description
default-properties	Optional	Specifies the default properties that apply to all admin objects (at the global level) in the resource adapter. The default-properties element can contain one or more property elements, each holding a name and value pair. See "properties" on page A-13.
admin-object-group	One or more	Specifies the default parameters that apply to all admin objects belonging to a particular admin object group specified in the ra.xml deployment descriptor. The properties specified in the group override any parameters that are specified at the global level. For admin-object-group subelements, see "admin-object-group" on page A-15.

admin-object-group

The admin-object-group element is used to define an admin object group. At the group level, you specify parameters that apply to all admin objects belonging to a particular admin object group specified in the ra.xml deployment descriptor. The properties specified in the group override any parameters that are specified at the global level.

The admin-object-interface element (a subelement of the admin-object-group element) serves as a required unique element (a key) to each admin-object-group. There must be a one-to-one relationship between the weblogic-ra.xml admin-object-interface element and the ra.xml adminobject-interface element

The admin-object-group element is a sub-element of the weblogic-connector element. You can define the following elements within the admin-object-group element.

Table 9-10 admin-object-group

Element	Required Optional	Description
admin-object-interface	Required	The admin-object-interface element serves as a required unique element (a key) to each admin-object-group. There must be a one-to-one relationship between the weblogic-ra.xml admin-object-interface element and the ra.xml adminobject-interface element.
default-properties	Optional	Specifies all the default properties that apply to all admin objects in this admin object group. The default-properties element can contain one or more property elements, each holding a name and value pair. See "properties" on page A-13.
admin-object-instance	One or more	Specifies one or more admin object instances within the admin object group, corresponding to the admin object instances for the resource adapter. You can specify properties at the instance level and override those provided in the group and global levels. For subelements, see "admin-object-instance" on page A-17.

admin-object-instance

You can define the following subelements under `admin-object-instance`.

Table 9-11 admin-object-instance subelements

Element	Required Optional	Description
`jndi-name / resource-link`	Required	The admin object group that defines the reference name for the admin object instance. You can specify the reference name to be the JNDI name or resource link of the connection instance. If the JNDI name is specified (by specifying the `jndi-name` element), the connection pool is bound into a JNDI that clients outside the application can see. **Note:** In order for this to work, the `enable-access-outside-app` element must be set to `true`. For resource adapters that do not need to be externally visible to other applications, you would specify the `resource-link` value.
`admin-object-properties`	Optional	Defines all the properties that apply to the admin object instance. The `admin-object-properties` element can contain one or more `property` elements, each holding a `name` and `value` pair. See "properties" on page A-13.

outbound-resource-adapter

The `outbound-resource-adapter` element is a sub-element of the `weblogic-connector` element. You can define the following elements within the `outbound-resource-adapter` element.

Table 9-12 outbound-resource-adapter subelements

Element	Required Optional	Description
`default-connection-properties`	Optional	This complex element is used to specify the properties at an global level. At this level, the user is able to specify parameters that apply to all outbound connection pools in the resource adapter. For subelements, see "default-connection-properties" on page A-19.
`connection-definition-group`	One or more	This element is used to specify all the connection defintion groups. There must be a one-to-one correspondence relationship between the connection factories in the `ra.xml` deployment descriptor and the groups in the `weblogic-ra.xml` deployment descriptor. A group does not have to exist in the `weblogic-ra.xml` deployment descriptor for every connection factory in `ra.xml`. However, if a group exists, there must be at least one connection instance in the group. The properties specified in the group override any parameters that are specified at the global level using `default-connection-properties`. For subelements, see "connection-definition-group" on page A-28.

default-connection-properties

The `default-connection-properties` element is a sub-element of the `outbound-resource-adapter` element. You can define the following elements within the `default-connection-properties` element.

Table 9-13 default-connection-properties subelements

Element	Required Optional	Description
`pool-params`	Optional	Serves as the root element for providing connection pool-specific parameters for this connection factory. WebLogic Server uses these specifications to control the behavior of the maintained pool of ManagedConnections. This is an optional element. Failure to specify this element or any of its specific element items results in default values being assigned. Refer to the description of each individual element for the designated default value. For subelements, see "pool-params" on page A-22.
`logging`	Optional	Contains parameters for configuring logging of the `ManagedConnectionFactory` and `ManagedConnection` objects of the resource adapter. For subelements, see "logging" on page A-25.

Table 9-13 default-connection-properties subelements

Element	Required Optional	Description

`transaction-support`	Optional	Specifies the level of transaction support for a particular Connection Factory. It provides the ability to override the transaction-support value specified in the `ra.xml` deployment descriptor that is intended to be the default value for all Connection Factories of the resource adapter.
		
		The value of transaction-support must be one of the following:
		
		`NoTransaction`
		
		`LocalTransaction`
		
		`XATransaction`
		
		For related information, see Chapter 5, "Connection Management."

`authentication-mechanism`	Optional	The `authentication-mechanism` element specifies an authentication mechanism supported by a particular Connection Factory in the resource adapter. It provides the ability to override the `authentication-mechanism` value specified in the `ra.xml` deployment descriptor that is intended to be the default value for all Connection Factories of the resource adapter. Note that `BasicPassword` mechanism type should support the `javax.resource.spi.security.PasswordCredential` interface.

Table 9-13 default-connection-properties subelements

Element	Required Optional	Description
`reauthentication-support`	Optional	A Boolean that specifies whether a particular connection factory supports re-authentication of an existing `ManagedConnection` instance. It provides the ability to override the `reauthentication-support` value specified in the `ra.xml` deployment descriptor that is intended to be the default value for all Connection Factories of the resource adapter.
`properties`	Optional	The `properties` element includes one or more property elements, which define name and value subelements that apply to the default connections.
`res-auth`	Optional	Specifies whether to use container- or application-managed security. The values for this element can be one of `Application` or `Container`. The default value is `Container`.

pool-params

The `pool-params` element is a sub-element of the `default-connection-properties` element. You can define the following elements within the `pool-params` element.

Table 9-14 pool-params subelements

Element	Required Optional	Description
`initial-capacity`	Optional	Specifies the initial number of ManagedConnections, which WebLogic Server attempts to create during deployment. Default Value: 1
`max-capacity`	Optional	Specifies the maximum number of ManagedConnections, which WebLogic Server will allow. Requests for newly allocated ManagedConnections beyond this limit results in a `ResourceAllocationException` being returned to the caller. Default Value: 10
`capacity-increment`	Optional	Specifies the maximum number of additional ManagedConnections that WebLogic Server attempts to create during resizing of the maintained connection pool. Default Value: 1
`shrinking-enabled`	Optional	Specifies whether unused ManagedConnections will be destroyed and removed from the connection pool as a means to control system resources. Default Value: true
`shrink-frequency-seconds`	Optional	Specifies the amount of time (in seconds) the Connection Pool Management waits between attempts to destroy unused ManagedConnections. Default Value: 900 seconds

Table 9-14 pool-params subelements

Element	Required Optional	Description
`highest-num-waiters`	Optional	Specifies the maximum number of threads that can concurrently block waiting to reserve a connection from the pool. Default Value: 0
`highest-num-unavailable`	Optional	Specifies the maximum number of ManagedConnections in the pool that can be made unavailable to the application for purposes such as refreshing the connection. Note that in cases like the backend system being unavailable, this specified value could be exceeded due to factors outside the pool's control. Default Value: 0
`connection-creation-retry-frequency-seconds`	Optional	The periodicity of retry attempts by the pool to create connections. Default Value: 0
`connection-reserve-timeout-seconds`	Optional	Sets 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-seconds`	Optional	The frequency with which connections in the pool are tested. Default Value: 0
`test-connections-on-create`	Optional	Enables the testing of newly created connections. Default Value: false
`test-connections-on-release`	Optional	Enables testing of connections when they are being released back into the pool. Default Value: false

Table 9-14 pool-params subelements

Element	Required Optional	Description
`test-connections-on-reserve`	Optional	Enables testing of connections when they are being reserved. Default Value: false
`profile-harvest-frequency-seconds`	Optional	Specifies how frequently the profile for the connection pool is being harvested.
`ignore-in-use-connections-enabled`	Optional	When the connection pool is being shut down, this element is used to specify whether it is acceptable to ignore connections that are in use at that time.
`match-connections-supported`	Optional	Indicates whether 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. Default Value: true

logging

The logging element is a sub-element of the default-connection-properties element. You can define the following elements within the logging element.

Table 9-15 logging subelements

Element	Required Optional	Description
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.
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. Default Value: false

Table 9-15 logging subelements

Element	Required Optional	Description
rotation-type	Optional	Sets the file rotation type. Possible values are bySize, byName, none bySize—When the log file reaches the size that you specify in file-size-limit, the server renames the file as FileName.n. byName—At each time interval that you specify in file-time-span, the server renames the file as FileName.n. After the server renames a file, subsequent messages accumulate in a new file with the name that you specified in log-filename. none—Messages accumulate in a single file. You must erase the contents of the file if the log size becomes unwieldy. Default Value: bySize
number-of-files-limited	Optional	Specifies whether to limit the number of files that this server instance creates to store old log messages. (Requires that you specify a rotation-type of bySize). After the server reaches this limit, it overwrites the oldest file. If you do not enable this option, the server creates new files indefinitely and you must clean up these files as you require. If you enable number-of-files-limited by setting it to true, the server refers to your rotationType variable to determine how to rotate the log file. Rotate means that you override your existing file instead of creating a new file. If you specify false for number-of-files-limited, the server creates numerous log files rather than overriding the same one. Default Value: false

Table 9-15 logging subelements

Element	Required Optional	Description
file-count	Optional	The maximum number of log files that the server creates when it rotates the log. This number does not include the file that the server uses to store current messages. (Requires that you enable number-of-files-limited.) Default Value: 7
file-size-limit	Optional	The size that triggers the server to move log messages to a separate file. (Requires that you specify a rotation-type of bySize.) After the log file reaches the specified minimum size, the next time the server checks the file size, it will rename the current log file as FileName.n and create a new one to store subsequent messages. Default Value: 500
rotate-log-on-startup	Optional	Specifies whether a server rotates its log file during its startup cycle. Default Value: true
log-file-rotation-dir	Optional	Specifies the directory path where the rotated log files will be stored.

Table 9-15 logging subelements

Element	Required Optional	Description
rotation-time	Optional	The start time for a time-based rotation sequence of the log file, in the format k:mm, where k is 1-24. (Requires that you specify a rotation-type of byTime.) At the specified time, the server renames the current log file. Thereafter, the server renames the log file at an interval that you specify in file-time-span. If the specified time has already past, then the server starts its file rotation immediately. By default, the rotation cycle begins immediately.
file-time-span	Optional	The interval (in hours) at which the server saves old log messages to another file. (Requires that you specify a rotation-type of byTime.) Default Value: 24

connection-definition-group

The connection-definition-group element is used to define a connection definition group. At the group level, you specify parameters that apply to all outbound connections belonging to a particular connection factory specified in the ra.xml deployment descriptor using the connection-definition-group element. A one-to-one correspondence exists from a connection factory in ra.xml to a connection definition group in weblogic-ra.xml. The properties specified in a group override any parameters specified at the global level.

The connection-factory-interface element (a subelement of the connection-definition-group element) serves as a required unique element (a key) to each connection-definition-group. There must be a one-to-one relationship between the weblogic-ra.xml connection-definition-interface element and the ra.xml connectiondefinition-interface element.

The connection-definition-group element is a sub-element of the outbound-resource-adapter element. You can define the following elements within the connection-definition-group element.

Table 9-16 connection-definition-group subelements

Element	Required Optional	Description
connection-factory-interface		Every connection definition group has a key (a required unique element). This key is the connection-factory-interface. The value specified for connection-factory-interface must be equal to the value specified for connection-factory-interface in ra.xml.
default-connection-properties		This complex element is used to define properties for outbound connections at the group level. See "default-connection-properties" on page A-19.
connection-instance		Under each connection definition group, the user can specify connection instances. These correspond to the individual connection pools for the resource adapter. Parameters can be specified at this level too and these override those provided in the group and global levels. This element specifies a description of the connection pool. (A connection instance is equivalent to a connection pool.) It is used to document the connection pool. See "connection-instance" on page A-30.

connection-instance

You can define the following subelements under connection-instance.

Table 9-17 connection-instance subelements

Element	Required Optional	Description
description	Optional	Specifies a description of the connection instance.
jndi-name resource-link	Required	The connection definition group that defines the reference name for the connection instance. The reference name can be a JNDI name or a resource link.
connection-properties	Optional	Defines all the properties that apply to the connection instance. The connection-properties element can contain one or more property elements, each holding a name and value pair. See “properties” on page A-13.

Resource Adapter Best Practices

This appendix describes some best practices for resource adapter developers.

- "Classloading Optimizations for Resource Adapters" on page B-1
- "Connection Optimizations" on page B-2
- "Thread Management" on page B-2
- "InteractionSpec Interface" on page B-2

Classloading Optimizations for Resource Adapters

You can package resource adapter classes in one or more JAR files, and then place the JAR files in the RAR file. These are called nested JARs. When you nest JAR files in the RAR file, and classes need to be loaded by the classloader, the JARs within the RAR file must be opened and closed and iterated through for each class that must be loaded.

If there are very few JARs in the RAR file and if the JARs are relatively small in size, there will be no significant performance impact. On the other hand, if there are many JARs and the JARs are large in size, the performance impact can be great.

To avoid such performance issues, you can either:

1. Deploy the resource adapter in an exploded format. This eliminates the nesting of JARs and hence reduces the performance hit involved in looking for classes.

2. If deploying the resource adapter in exploded format is not an option, the JARs can be exploded within the RAR file. This also eliminates the nesting of JARs and thus improves the performance of classloading significantly.

Connection Optimizations

BEA recommends that resource adapters implement the optional enhancements described in sections 7.14.2 and 7.14.2 of the J2CA 1.5 Specification. Implementing these interfaces allows WebLogic Server to provide several features that will not be available without them.

Lazy Connection Association, as described in section 7.14.1, allows the server to automatically clean up unused connections and prevent applications from hogging resources. Lazy Transaction Enlistment, as described in 7.14.2, allows applications to start a transaction after a connection is already opened.

Thread Management

Resource adapter implementations should use the `WorkManager` (as described in Chapter 10, "Work Management" in the J2CA 1.5 Specification) to launch operations that need to run in a new thread, rather than creating new threads directly. This allows WebLogic Server to manage and monitor these threads.

InteractionSpec Interface

WebLogic Server supports the Common Client Interface (CCI) for EIS access, as defined in Chapter 15, "Common Client Interface" in the J 2CA 1.5 Specification. The CCI defines a standard client API for application components that enables application components and EAI frameworks to drive interactions across heterogeneous EISes.

As a best practice, you should not store the `InteractionSpec` class that the CCI resource adapter is required to implement in the RAR file. Instead, you should package it in a separate JAR file outside of the RAR file, so that the client can access it without having to put the `InteractionSpec` interface class in the generic CLASSPATH.

With respect to the `InteractionSpec` interface, it is important to note that when all application components (EJBs, resource adapters, Web applications) are packaged in an EAR file, all common classes can be placed in the `APP-INF/lib` directory. This is the easiest possible scenario.

This is not the case for standalone resource adapters (packaged as RAR files). If the interface is serializable (as is the case with `InteractionSpec`), then both the client and the resource adapter

need access to the `InteractionSpec` interface as well as the implementation classes. However, if the interface extends `java.io.Remote`, then the client only needs access to the interface class.