BEA WebLogic Server™

Programming WebLogic Resource Adapters

Version 9.0 BETA
Revised: December 15, 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

6. Transaction Management

7. Messaging and Transactional Inflow

8. Security

9. Packaging and Deploying Resource Adapters

A. weblogic-ra.xml Schema

B. Resource Adapter Best Practices

CHAPTER 1

Introduction and Roadmap

This section describes the contents and organization of this guide—*Programming WebLogic Resource Adapters*.

Document Scope and Audience

This document is a resource for software developers who develop applications that include J2EE resource adapters to be deployed to WebLogic Server. It is also a resource for resource adapter users and deployers. 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 1-2.

It is assumed that the reader is familiar with J2EE and resource adapter concepts. 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 chapter, 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, "Configuration Tasks," outlines configuration requirements for the BEA WebLogic Server implementation of the J2EE Connector Architecture.
- Chapter 5, "Connection Management,"introduce you to resource adapter connection management:
- Chapter 6, "Transaction Management," introduce you to the resource adapter transaction management:
- Chapter 7, "Messaging and Transactional Inflow," introduces you to resource adapter messaging inflow and transactional inflow.
- Chapter 9, "Packaging and Deploying Resource Adapters," discusses packaging and deploying requirements for resource adapters and provides instructions for performing these tasks.
- Chapter 8, "Security," discusses WebLogic Server resource adapter security:
- 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

This document contains resource adapter-specific design and development information.

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

- *Developing WebLogic Server Applications* is a guide to developing WebLogic Server applications.
- *Deploying WebLogic Server Applications* 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.
- Sun Microsystems J2EE Connector Architecture Specification, Version 1.5 Final Release (referred to as the J2CA 1.5 Specification in this guide). http://java.sun.com/j2ee/connector/

Examples for the Resource Adapter Developer

In addition to this document, BEA Systems provides two resource adapter examples for software developers within the context of the Avitek Medical Records Application (MedRec) sample, discussed in the next section.

Avitek Medical Records Application (MedRec)

MedRec is an end-to-end sample J2EE application shipped with WebLogic Server that simulates an independent, centralized medical record management system. The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients.

MedRec demonstrates WebLogic Server and J2EE features, and highlights BEA-recommended best practices. MedRec is included in the WebLogic Server distribution, and can be accessed from the Start menu on Windows machines. For Linux and other platforms, you can start MedRec from the `WL_HOME\samples\domains\medrec` directory, where `WL_HOME` is the top-level installation directory for WebLogic Platform.

Resource Adapter Examples in the WebLogic Server Distribution

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.

BEA provides the two resource adapter examples with this release of WebLogic Server. Both are compliant with the J2CA 1.5 Specification. BEA recommends that you run these resource adapter examples before developing your own resource adapters.

CHAPTER 2+

Understanding Resource Adapters

This section introduces you to the BEA WebLogic Server implementation of the J2EE Connector Architecture as well as the resource adapter types and XML schema.

Overview of the BEA WebLogic Server Implementation of the J2EE Connector Architecture

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

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

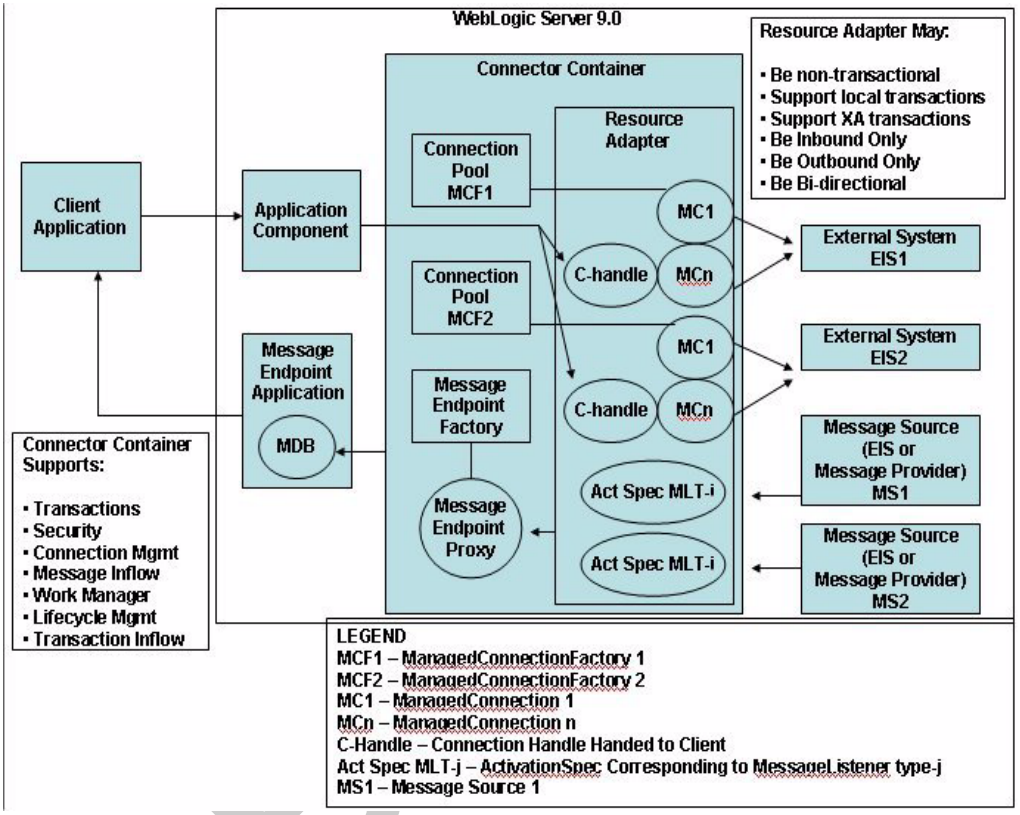
The resource adapter serves as a protocol adapter that allows any arbitrary EIS communication protocol to be used for connectivity. An application server vendor extends its system once to support the J2EE Connector Architecture and is then assured of seamless connectivity to multiple EISs. Likewise,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

J2EE Connector Architecture

Multiple resource adapters are pluggable into an application server. This capability enables application components deployed on the application server to access the underlying EISs. An application server and an EIS collaborate to keep all system-level mechanisms—transactions, security, and connection management—transparent from the application components. As a result, an application component provider focuses on the development of business and presentation logic for its 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 EISs.

For more information on the J2EE Connector Architecture, see Section 3 “The Connector Architecture” of the J2CA 1.5 Specification.

Figure 2-1 Connector Architecture Overview



Architecture Components

Figure 2-1, "Connector Architecture Overview," on page 2-3 contains the following components:

- A client application
- 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 (an 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 Connector container in which the resource adapter is deployed. The container contains the following:

  - A deployed resource adapter that provides bi-directional (inbound and outbound) communication to and from the EIS
  - 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 specs corresponding to a specific message listener type MLT-j
  - One of the connection pools maintained by the container for the management of 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 EISs)
  - 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 (MS1, MS2), which could be an Enterprise Information System (EIS) or Message Provider.

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. The EIS side of these system-level contracts are implemented in a resource adapter. 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—allows a secure access to an EIS and provides support for a secure application environment that reduces security threats to the EIS and protects valuable information resources managed by the EIS.
- Lifecycle management contract—allows WebLogic Server to manage the lifecycle of a resource adapter. Allows for bootstrapping of a resource adapter instance during its deployment or application server startup, and to notify the resource adapter instance during its undeployment or during an orderly shutdown of the application server.
- 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 allowing 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.

WebLogic Server Resource Adapters vs. WebLogic Integration Resource Adapters

It is important to note here the difference between BEA WebLogic Integration (WLI) resource adapters and BEA WebLogic Server resource adapters. WLI resource adapters are written to be specific to WebLogic Server, and in general, are not deployable to other application servers. However, 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*.

1.0 vs. 1.5 Resource Adapters

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. The capabilities provided by and for a 1.5 resource adapter may include:

- Outbound communication from the application to multiple systems, such as Enterprise Information Systems (EISs) and databases. See "Resource Adapter Types" on page 2-6.
- Inbound communication from an external system such as an EIS to the resource adapter. See "Handling of 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 "Supported Transaction Levels" on page 6-2.
- An application server-provided WorkManager to enable resource adapters to create threads through Work instances. See "work-manager" on page A-6.
- A lifecycle contract for start() and stop() of the resource adapter by the application server. See "Configuring the ra.xml File" on page 3-5.

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 a 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.

For more information on WebLogic Server resources adapters that are based on the J2EE 1.0 Connector Architecture, see *BEA WebLogic Adapter 8.1 Documentation*.

Resource Adapter Types

WebLogic Server supports three types of resource adapters:

- Outbound resource adapter—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 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. Only one outbound connection can be created for these resource adapters.

- Inbound resource adapter—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 "Transactional Inflow" on page 7-8.

  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 adapters—supports both outbound and inbound communication.

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:

- `ra.xml`—The standard J2EE deployment descriptor. All resource adapters must be specified in an `ra.xml`. See `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. See Appendix A, "weblogic-ra.xml Schema."

CHAPTER 3

Configuration Tasks

The following sections outline configuration tasks for WebLogic Server resource adapters:

Configuring Resource Adapters

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

Resource Adapter Overview

The J2EE 1.5 Connector Architecture enables both Enterprise Information System (EIS) vendors and third-party application developers to develop resource adapters that can be deployed in any application server supporting the Sun Microsystems J2EE Platform Specification, Version 1.4.

The resource adapter is the central piece in the J2EE 1.5 Connector Architecture; it serves as the *connector* between the client component and the EIS. When a resource adapter is deployed in the WebLogic Server environment, it enables the development of robust J2EE Platform applications that can access remote EIS systems and also receive messages and transactions from remote EIS systems. Resource adapters contain the Java components, and if necessary, the native components required to interact with the EIS.

Creating and Modifying Resource Adapters: Main Steps

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

Creating a New Resource Adapter Archive (RAR)

The following are the main steps for creating a resource adapter archive (RAR):

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

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

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

- `<connection-interface>java.sql.Connection</connection-interface>`
- `<connection-impl-class>com.sun.connector.blackbox.JdbcConnection</connection-impl-class>`

2. Compile the Java code using a standard compiler for the interfaces and implementation into class files.
3. Create the resource adapter-specific deployment descriptors:
   - `ra.xml` describes the resource adapter-related attributes type and its deployment properties using a standard DTD from Sun Microsystems.
   - `weblogic-ra.xml` adds additional WebLogic Server-specific deployment information.

   For detailed information about creating resource adapter-specific deployment descriptors, refer to Appendix A, "weblogic-ra.xml Schema."
4. Package the Java classes into a Java archive (JAR) file.

   The first step in creating a JAR file is to 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 you create the resource adapter archive by executing a `jar` command similar to the following in the staging directory:

   ```
   jar cvf myRAR.rar *
   ```
5. Auto-deploy the resource adapter archive file (RAR) on WebLogic Server for testing purposes.

   While you are testing the resource adapter, you might need to edit the resource adapter deployment descriptors. You can do this manually using a text editor. You can also edit a limited number of deployment descriptors using the Configuration tabs of the WebLogic Server Console.

   See Appendix A, "weblogic-ra.xml Schema,"for detailed information on the elements in these deployment descriptors.
6. Deploy the RAR resource adapter archive file on WebLogic Server or include it in an enterprise archive (EAR) file to be deployed as part of an enterprise application.

   Refer to *Deploying WebLogic Server Applications* for detailed information about deploying components and applications.

Modifying an Existing Resource Adapter (RAR)

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

1. Create a temporary directory 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 files: `Manifest.mf` and `ra.xml`

   Execute these commands to see these files:

   ```
   c:/stagedir> ls
   ```

   ```
          blackbox-notx.rar
   ```

   ```
          META-INF
   ```

   ```
   c:/stagedir> ls META-INF
   ```

   ```
          Manifest.mf
   ```

   ```
          ra.xml
   ```

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.

   Refer to Appendix A, "weblogic-ra.xml Schema," for more information on the `weblogic-ra.xml` XSD.

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.

Configuring the ra.xml File

If you do not have an `ra.xml` file, you must manually create or edit an existing one to set the necessary deployment properties for the resource adapter. You can use a text editor to edit the properties. For information on creating an `ra.xml` file, refer to the 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 `.xml` extensions for EJBs and Web applications in WebLogic Server, which also add WebLogic-specific deployment descriptors to the deployable archive. As is, the basic RAR or deployment directory can be deployed to WebLogic Server, but it will not provide available features and capabilities unless you first configure these. You create and configure WebLogic Server-specific deployment properties in the `weblogic-ra.xml` file and add that file to the deployment.

For example, you can configure the following attributes for each connection instance. For more information on connection properties, see Chapter 5, “Connection Management.”

- 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 WebLogic Server attempts to allocate at deployment time.
  - Maximum number of ManagedConnections WebLogic Server allows to be allocated at any one time.
  - Number of ManagedConnections 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.
- File to store logging information for the ManagedConnectionFactory or ManagedConnection.
- The amount of time a connection can remain idle.

Manually Editing XML Deployment Files

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

Basic Conventions

To manually edit XML elements:

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

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.

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

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: Most browsers do not display the contents of files having the `.xsd` extension. 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

Using the Administration Console Configuration tabs to view, modify, and (when necessary) persist deployment descriptor elements. Some of the descriptor element changes take place dynamically at runtime without requiring the resource adapter to be redeployed. Other descriptor elements will require the resource adapter to be redeployed.

Pool Parameters

Using the Configuration tabs, you can dynamically update the following `weblogic-ra.xml` pool parameters:

- `initial-capacity`
- `max-capacity`
- `capacity-increment`
- `shrinking-enabled`
- `shrink-frequency-seconds`
- `highest-num-waiters`
- `highest-num-unavailable`
- `connection-creation-retry-frequency-seconds`
- `connection-reserve-timeout-seconds`
- `test-frequency-seconds`
- `test-connections-on-create`
- `test-connections-on-release`
- `test-connections-on-reserve`
- `profile-harvest-frequency-seconds`

Logging Parameters

Using the Configuration tabs, you can dynamically update the following `weblogic-ra.xml` logging parameters:

- `log-filename`
- `logging-enabled`
- `rotation-type`
- `number-of-files-limited`
- `file-count`
- `file-size-limit`
- `rotate-log-on-startup`
- `log-file-rotation-dir`
- `rotation-time`
- `file-time-span`

Automatic Generation of the weblogic-ra.xml File

In WebLogic Server, a resource adapter archive (RAR) must include a `weblogic-ra.xml` deployment descriptor file in addition to the `ra.xml` deployment descriptor file specified in the 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 MySpecialRA.rar, the JNDI name of the connection factory will be eis/MySpecialRA.

For a 1.5 resource adapter having a ResourceAdapter bean class specified, the JNDI name of the bean would beMySpecialRA. 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 the current release of WebLogic Server, the Link-Ref mechanism has been deprecated and is now 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 Application Libraries and Optional Packages."

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

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

Do one of the following:

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

For instructions on editing the `weblogic-ra.xml` file, see Appendix A, "weblogic-ra.xml Schema."

CHAPTER 4

Programming Tasks

The following sections outline programming tasks for the WebLogic Server resource adapters

Programming a Resource Adapter to Perform as a Startup Class

You can program a resource adapter that has a minimal resource adapter class (a class that implements `javax.resource.ResourceAdapter`), which defines a `start()` and `stop()` method.

Note: Due to the definition of the `ResourceAdapter` interface, the `endpointActivation()/Deactivation()` and `getXAResource()` methods must also be defined.

You can use this type of resource adapter as an alternative to a WebLogic Server startup class. 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.

Since resource adapters have access to the Work Manager passed into them through the `BootstrapContext` in the `start()` method, they should submit Work instances instead of using direct Thread management. This enables WebLogic Server to effectively manage threads 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 printIn's). 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 uses the Work Manager and submits work instances. 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;
    }
}
}
```

Resource Adapter Suspend() and Resume()

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 could queue up all incoming messages while allowing in-flight transactions to complete, or you could notify the Enterprise Information System (EIS) that messages are temporarily not being received.

You would 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 reenabled. 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 or from the WebLogic Server Administration Console.

Methods exist for determining whether a resource adapter supports a particular type of suspension. Also a method exists to determine 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 redeploment sequence of calls are 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 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.

Diagnostic Context ID

With the new WebLogic Server diagnostic framework, a thread may have an associated *diagnostic context*. A request on the thread carries the 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 may be used, for example, to trace the execution from an EIS all the way in to a message endpoint. This capability may be used for a variety of diagnostic purposes. The String may be set, for example, to the client ID or session ID on an inbound message from an EIS. During message dispatch, various diagnostics can be gathered to see the request flow through the system. The `setDiagnosticContextID()` & `getDiagnosticContextID()` methods are provided for this purpose.

Dye Bits

Also available with the new diagnostic framework is the ability to *dye* a request . The ExtendedBootstrapContext allows for the setting and retrieval of 4 dye bits on the current thread for whatever diagnostic purpose the resource adapterdeveloper chooses. For example, the priority of a request may be set using the dye bits.

Additional ExtendedBootstrap Capabilities

There is also a `complete()` method on the ExtendedBootstrapContext. This is used as a callback to the connector contain. For detailed information on this feature, see "Production Redeployment" in *Deploying WebLogic Server Applications*.

Connection Management

The following sections introduce you to the WebLogic Server resource adapter connection management. For more information on connection management, see chapter 6 “Connection Management” of the J2CA 1.5 Specification.

Connection Management

This section discusses the connection management contract between WebLogic Server and a resource adapter. The connection management contract:

- 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.
- Provides a generic mechanism by which an application server can provide different services—transactions, security, advanced pooling, error tracing/logging—for its configured set of resource adapters.
- Provides support for connection pooling.

Connection Factory and Connection

An application component uses a connection factory, a public interface, 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 acts as a factory of connections, providing connection and connection factory interfaces. A connection factory acts as a factory for EIS connections. For example, `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 5-3.

Resource Adapters Bound in JNDI Tree

Version 1.0 resource adapters are identified by their ConnectionFactory objects bound in the JNDI tree. Version 1.5 resource adapters can now 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).

At deployment time, the ResourceAdapter Bean (if it exists) is bound into the JNDI tree using the value that is set for 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 publically exposed by the resource adapter provider. (To set the `jndi-name` value, see Appendix A, "weblogic-ra.xml Schema.")

Obtaining the ConnectionFactory (Client-JNDI Interaction)

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 tasks are performed 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 EISs.

1. The application assembler or component provider specifies the connection factory requirements for an application component by using a deployment descriptor mechanism. For example:
   - `res-ref-name: eis/myEIS`
   - `res-type: javax.resource.cci.ConnectionFactory`
   - `res-auth: Application or Container`
2. The resource adapter deployer sets the configuration information for the resource adapter.
3. The application server uses a configured resource adapter to create physical connections to the underlying EIS.
4. The application component looks up a connection factory instance in the component's environment by using the JNDI interface.

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 a connection factory instance of type `java.resource.cci.ConnectionFactory` as specified in the `res-type` element.

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

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

6. The application component uses the returned connection to access the underlying EIS.
7. After the component finishes with the connection, it closes the connection using the close method on the `Connection` interface.

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

8. If an application component fails to close an allocated connection after its use, that connection is considered an unused connection. The application server manages the cleanup of unused connections.

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. WebLogic Server allows you to set outbound connection properties through the use of the `outbound-resource-adapter` element in the `weblogic-ra.xml` deployment descriptor.

Connection Pool Configuration Levels

The `outbound-resource-adapter` element in the `weblogic-ra.xml` deployment descriptor is used to describe the outbound components of a resource adapter.

There are three levels for defining outbound connection pools:

1. Global level—at this level, you 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.

2. Group level—at this level, you 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-29.

   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

3. The instance level—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. Properties can be specified at this level too and these override those provided at the group and global levels. See "connection-instance" on page A-30.

Multiple Outbound Connections Example

The following is an example of a `weblogic-ra.xml` schema that has multiple outbound connections configured:

Listing 5-2 weblogic-ra.xml Schema: 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-n
ame>
<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-second
        s>
    </pool-params>
    <properties>
        <property>
            <name>ConnectionURL</name>
            <value>jdbc:oracle:thin:@bcpdb:1531:bay920;create
            =true;autocommit=false</value>
        </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-n
    ame>
    <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-second
            s>
        </pool-params>
        <properties>
            <property>
                <name>ConnectionURL</name>
                <value>jdbc:oracle:thin:@bcpdb:1531:bay920;create
                =true;autocommit=false</value>
            </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</connect
    ion-factory-interface>
    <connection-instance>
        <jndi-name>eis/900eisaBlackBoxXATxConnectorJNDINAME3</jndi-n
        ame>
        <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-second
                s>
            </pool-params>
            <properties>
                <property>
                    <name>ConnectionURL</name>
```

```
                <value>jdbc:oracle:thin:@bcpdb:1531:bay920;create
                =true;autocommit=false</value>
            </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>
```


Support of Inbound Connections

In previous releases of the J2EE Connector Architecture, you could configure a single outbound connection and no inbound connections. You can now configure inbound resource adapters based on the J2CA 1.5 Specification.

The following are the main steps involved in configuring an inbound connection:

1. Configure and create a new resource adapter as discussed in Chapter 3, "Configuration Tasks."
2. Provide a JNDI name for the resource adapter in the `weblogic-ra.xml` deployment descriptor. See "jndi-name" on page A-2.
3. Within the packaged enterprise application, include a configured EJB message-driven bean (MDB). For the MDB, provide the same JNDI name assigned to the resource adapter in the previous step. You do so by setting the this value in the `resource-adapter-jndi-name` element in the `weblogic-ejb-jar.xml` deployment descriptor. 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 8-7.
5. Deploy the resource adapter as discussed in *Deploying WebLogic Server Applications*.
6. Deploy the MDB as discussed in *Deploying WebLogic Server Applications*.

The following example shows an inbound connection configuration with two message listener/activation specs:

Listing 5-3 Configuring an Inbound Connection Example

```
<inbound-resourceadapter>
    <messageadapter>
        <messagelistener>
            <messagelistener-type>weblogic.qa.tests.connector.adapters.flex
            .InboundMsgListener</messagelistener-type>
            <activationspec>
                <activationspec-class>weblogic.qa.tests.connector.adapters.f
                lex.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.f
                lex.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. 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. As a result, you may decide to populate the connection pool with an initial number of ManagedConnections upon startup of WebLogic Server and therefore avoid creating them at run time. You can configure this setting using the `initial-capacity` element in the `weblogic-ra.xml` descriptor file. The default value for this element is `1` ManagedConnection.

Since no initiating security principal or request context information is known at WebLogic Server startup, initial connections are created using a security subject that may be specified by the `security` element, described in "Configuring Security Identities for Resource Adapters" on page 8-7.

max-capacity: Setting the Maximum Number of ManagedConnections

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

You configure this setting using the `max-capacity` element in the `weblogic-ra.xml` descriptor file. If a new ManagedConnection (or more than one ManagedConnection in the case of `capacity-increment` being greater than one) needs to be created during a connection request, WebLogic Server ensures that no more than the maximum number of allowed ManagedConnections are created. 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 any existing and available ManagedConnection in the connection pool. However, if a match is not found, a new ManagedConnection may be created to satisfy the connection request.

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

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 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 Clients Waiting for a Connection

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

If the connection fails while creating additional ManagedConnections, you can configure Weblogic Server to attempt to recreate it using the `connection-creation-retry-frequency-seconds` elements. By default, this feature is disabled.

connection-reserve-timeout-second: Reserving Connections

The `connection-reserve-timeout-seconds` element specifies the amount of time to wait for a reserved connection call. If the time has been exceeded, the caller encounters an exception. By default, the value of this element is `-1`, which means that by default this element is set to not wait for a reserved connection.

When a call to reserve a connection is made, WebLogic Server attempts to obtain a connection from the pool of available connections. If none are available, it tries to create new connections using the `capacity-increment` element and then attempts to obtain a connection from the pool of newly created connections.

match-connections-supported: Matching Connections

When a connection request is made, the request contains parameter information. By default, the connector container calls the `matchManagedConnections()` method on the ManagedConnection factory 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 ManagedConnection factory 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 ManagedConnection factory.

You can set the `match-connections-supported` element to specify whether the 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 has not been reached or 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 the periodicity (in seconds)at which 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.

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.

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.

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 customer (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 1.0 resource adapters. For 1.0 resource adapters, continue to use the 8.1 deployment descriptors found in `weblogic-ra.xml`. It contains elements that continue to accomodate 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

The J2CA 1.5 Specification allows an application server to test the validity of existing connections through the ValidatingManagedConnectionFactory interface, which the

ManagedConnectionFactory may implement. 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.

The WebLogic Server connector container takes advantage of ManagedConnectionFactories that implement this interface to test connections in the pool. The `weblogic-ra.xml` schema provides the following elements, which allow you to control the testing of connections in the pool.

Note: If the ManagedConnectionFactory does not implement the ValidatingManagedConnectionFactory interface, the test functionality is not valid.

- `test-frequency-seconds`—The connector container periodically tests all the free connections in the pool. This element is used to test the frequency with which the connections are tested. This is an optional parameter and by default it is set to 0, which means the connections will not be tested.
- `test-connections-on-create`—This element is used to determine if the connection should be tested upon its creation. It is an optional parameter and by default it is false.
- `test-connections-on-release`—This element is used to determine if the connection should be tested upon its release. It is an optional parameter and by default it is false.
- `test-connections-on-reserve`—This element is used to determine if the connection should be tested upon its reserve. It is an optional parameter and by default it is false.

The Admin Console provides the capability to test a resource adapter's connection pools. To do so:

1. Select the resource adapter in the Deployments table.
2. Select the Test tab.
3. You will see a table of connection pools for the resoure adapter and the test status of each pool.

For more information on this feature, see section 6.5.3.4 “Detecting Invalid Connections” in the J2CA 1.5 Specification.

Transaction Management

This section discusses the transaction management contract between WebLogic Server and an EIS resource manager. For more information on transaction management, see chapter 7 “Transaction Management” of the J2CA 1.5 Specification.

This section discusses the system-level transaction management contract that is used for outbound communication from WebLogic Server to Enterprise Information Systems (EISs). It also discusses the system-level transaction inflow contract that is used for inbound communication from EISs to WebLogic Server.

Supported Transaction Levels

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

WebLogic Server utilizes the WebLogic Server Transaction Manager implementation and supports resource adapters having the following transaction support levels:

- XA transaction support—allows a transaction to be managed by a transaction manager external to a resource adapter (and therefore external to an EIS). A resource adapter defines the type of transaction support by specifying the transaction-support element in the `ra.xml` file; a resource adapter can only support one type. When an application component demarcates an EIS connection request as part of a transaction, the application server is responsible for enlisting the XA resource with the transaction manager. When the application component closes that connection, the application server cleans up the EIS connection once the transaction has completed.

- Local transaction support—allows WebLogic Server to manage resources that are local to the resource adapter. Unlike XA transaction, local transaction cannot participate in a two-phase commit protocol (2PC).

  There may be one local transaction resource adapter involved with no transaction support resources in a 2PC transaction. 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, as described here. A resource adapter defines the type of transaction support by specifying the transaction-support element in the resource adapter `ra.xml` file; this is configured per connection instance.

  You can use the transaction-support element in the `weblogic-ra.xml` deployment descriptor to override the value specified in `ra.xml`. See "transaction-support" on page A-20 for details.

  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, which is configured to use local transaction support, is made 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 may 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.

Specifying the Transaction Levels in the RAR Configuration

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>`

You can override the transaction level type value specified in the `ra.xml` deployment descriptor that is intended to be the default value for all Connection Factories of the resource adapter. You do so by specifying a value in the `transaction-support` element of the `weblogic-ra.xml` deployment descriptor. The `transaction-support` element specifies the level of transaction support for a particular Connection Factory.

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

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

For more information on specifying the transaction level in the RAR configuration, see Section 17.6 “Resource Adapter XML Schema Definition” of the J2CA 1.5 Specification.

CHAPTER 7

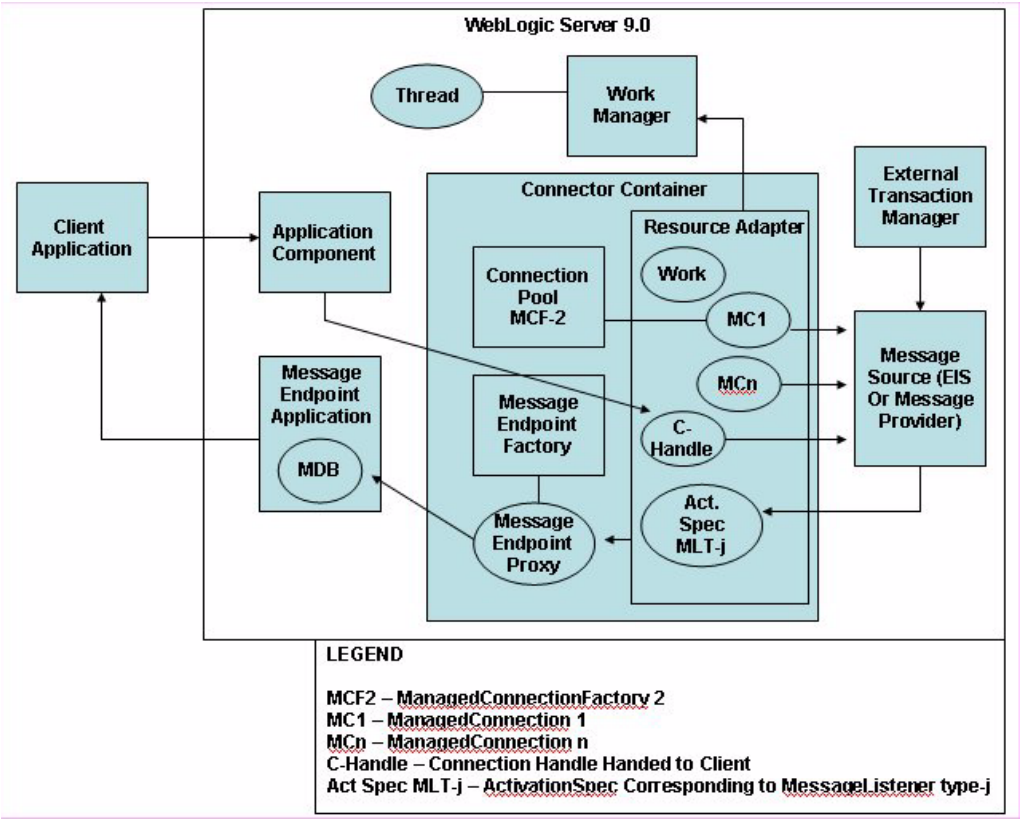
Messaging and Transactional Inflow

This section discusses resource adapter messaging inflow and transactional inflow.

Messaging and Transactional Inflow Architecture

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, "Messaging and Transactional Inflow Architecture," on page 7-2 contains the following components:

- A client application
- 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 (an 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, which is either subordinate to the WebLogic Server Transaction Manager for outbound XA transactions to the EIS or 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 specs corresponding to specific message listener types, MLT-j
  - One of the connection pools maintained by the container for the management of 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 EISs)
  - 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 (MS1, MS2), which could be an Enterprise Information System (EIS) or Message Provider.

Scenarios

There are two basic communication scenarios that may be described using the diagram: outbound and inbound communication. For related information, see Figure 2-1, “Connector Architecture Overview,” on page 2-3.

Outbound Sequence

A typical simplified outbound sequence involves the following steps:

1. The client application looks up the application component (EJB) in the JNDI that it wants to access.
2. The application component looks up the connection factory for the resource adapter in JNDI.
3. A getConnection() call is made on the connection factory causing a number of things to happen including:
   a. The resource adapter asks the container for an available managed connection from the pool and the container either returns an existing one or asks the resource adapter (through the ManagedConnectionFactory) to create a new one.
   b. The resource adapter creates a connection handle (C-handle) and returns it to the client application.
4. The client application invokes a request method on the application component.
5. The application component proceeds to make calls on a connection handle to process the client request (either using a cached connection handle or creating a new one as described above).
6. Using the managed connection corresponding to the connection handle, the resource adapter sends the request to the EIS.
7. The result of the request is returned back to the application component and ultimately to the client application using the connection handle.

Inbound Sequence

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 to perform the succeeding work in a separate Thread and 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 an MDB 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 MDB may handle the message directly and possibly return a result back to the EIS through the resource adapter, or it may distribute the message to some other application component, place the message on a queue to be picked up by the client or directly communicate with the client application.

Message Inflow Overview

A resource adapter that provides for message inflow typically includes the following:

- A proprietary communications channel and protocol for connecting to and communicating with an Enterprise Information System (EIS). (This communications channel and protocol are opaque 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 of Inbound Messages

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

- Handle it locally, that is, within the ResourceAdapter bean, without involving other components.
- Pass it off to another application component. For example, it may look up an EJB and invoke a method on it.

- Send it to a message endpoint. Typically a message endpoint is a Message-driven Bean (MDB).

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. The sender may not expect a response back as part of the same exchange with the resource adapter; this is called asynchronous or event notification-based coummunication. 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 non-transactional or XA. See Chapter 6, "Transaction Management."

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 7-8.

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 may expose connection configuration information to the deployer through a variety of means. This may be, for example, as properties on the ResourceAdapter bean or properties on the ActivationSpec object. An alternative would be 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) only supported Java Message Service (JMS) messaging. That is, an MDB had to implement the javax.jms.MessageListener interface that required the onMessage(javax.jms.Message) message listener method to be implemented. 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 listner interface.
- A deployment-time binding between an MDB and a resource adapter.

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 applciation (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.

How to Bind 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 with which the MDB will be associated. 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.

The Sequence of Events

In order to illustrate what occurs next, we will assume that the resource adapter is deployed prior to the MDB. However, the reverse order is possible; the deployed MDB polls until the resource adapter is deployed. Upon deployment, the following occurs:

1. The resource adapter is deployed and the ResourceAdapter bean is bound into JNDI using the name specified.

2. 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.
3. The MDB container provides the resource adapter with a configured ActivationSpec (containing configuration information) and a factory for the creation of message endpoint instances.
4. The resource adapter saves this information for later use in message delivery. The resource adapter knows what message listener interface the MDB implements. This information is important for determining what kind of messages to deliver to the MDB.

Dispatching of 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 listner 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.

Transactional Inflow

This section discusses the transaction inflow between WebLogic Server or an EIS and a resource adapter. A transaction inflow contract allows the resource adapter to flow-in 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.

As part of message inflow or WorkRequests, an EIS may “flow in” a transactional context under which messages are delivered or work is performed. The inbound transaction will be controlled by an external transaction manager. 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 foreign Transaction Manager.

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

The WebLogic Server Transaction Manager is subordinate to the foreign Transaction Manager. See “Overview of Participating in Foreign-Managed Transactions.”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 sequence of steps illustrates how a resource adapter would participate in a foreign 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 foreign transaction ID.
2. The resource adapter decodes the foreign 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 from above 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 foreign 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 foreign Transaction Manager and the WebLogic Server Transaction Manager, which is acting as a resource manager.

Using the Transactional Inflow Model for Locally Managed Transactions

As the resource adapter developer, if you receive requests from application components running in the same server instance as the resource adapter needing 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 (as discussed in "Transactional Inflow" on page 7-8).

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

CHAPTER 8

Security

The following sections discuss WebLogic Server resource adapter security:

Container-Managed and Application-Managed Sign-on

As specified in the J2CA 1.5 Specification, WebLogic Server supports both container-managed and application-managed sign-on. At runtime, WebLogic Server determines—based upon the specified information in the invoking client component's deployment descriptor—the chosen sign-on mechanism. This may also be specified in the `res-auth` element within the `weblogic-ra.xml` deployment descriptor. The element in the descriptor takes precedence over the calling components.

If the Weblogic Server J2EE 1.5 Connector Architecture implementation is unable to determine what sign-on mechanism is being requested by the client component, the connector container attempts container-managed sign-on.

It is important to note that even when using container-managed sign-on, if the client component has specified explicit security information, this information is also presented on the call to obtain the connection.

Application-Managed Sign-on

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

Container-Managed Sign-on

To use container-managed sign-on, WebLogic Server must identify a resource principal and then request the connection on behalf of the resource principal. In order to make this identification, WebLogic Server looks for a configured mapping in the embedded LDAP storage. For any deployed resource adapter, you can configure credential mappings for applicable users. You map a user in WebLogic Server to an appropriate set of credentials for a given resource adapter. 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 `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. Therefore, WebLogic Server can authenticate the user without compromising the confidentiality of the user's ID and password.

For more information, see "Configuring SSL" in *Managing WebLogic Security*:

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 you to provide 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. In this case, BEA recommends that—for the initial installation and deployment of this resource adapter—you set the `initial-capacity` to `0` for its connection pool. Once you have configured the appropriate

credentials and after the initial deployment of the resource adapter, you can change the `initial-capacity` element. For more information on `weblogic-ra.xml` deployment descriptors, see Appendix A, "weblogic-ra.xml Schema."

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

1. It searches for specific mappings at the connection factory level.
2. It searches for specific mappings at the global level.
3. It searches for default mappings at the connection factory level.
4. It searches for default mappings at the global level.

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

Listing 8-1 Example Mappings

```
pool1
system user name: admin
system password: adminpw
default user name: guest1
default password: guest1pw1

pool2
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 `pool1`. Since a credential mapping is defined for `system` for `pool1`, the application uses this mapping (`admin/adminpw`).

If the application makes the same request against `pool2` as `system`, it does not find the mapping and then searches for the mapping at the `global` level where it finds a mapping (`sysman/sysmanpw`).

If another application authenticates as `wlsjoe` and makes a request against `pool1`, it finds no mapping for `wlsjoe` defined for `pool1`. It then searches at the global level and finds a mapping for `wlsjoe` (`scott/tiger`). Against `pool2`, the application would find the mapping defined for `pool2` (`harry/harrypw`).

If an application authenticated as `user1` makes a request against `pool1`, it finds no mapping for `user1` for `pool1`. The following sequence occurs:

1. The application searches at the global level, which also has no mapping user1.
2. The application searches the pool1 mappings for a default mapping and finds a default mapping.

Creating Credential Mappings Using the Console

Credential maps can be created through the WebLogic Server Administration Console. If you are using the WebLogic Credential Mapping provider, the credential maps are stored in the embedded LDAP server.

To create a credential map:

1. Verify the Ignore Security Data in Deployment Descriptors attribute is enabled on the default (active) security realm. Otherwise, you risk overwriting credential maps with old information in weblogic-ra.xml deployment descriptor files.
2. Define a user or group for the EIS user. For more information, see "Defining Users and Groups" on page 8-6.
3. Deploy a resource adapter.

4. In the left pane of the WebLogic Server Administration Console, expand Deployments, then select the resource adapter.
5. Select the Security tab and then the Credential Mapping subtab.
6. Click the New button.
7. Resource adapters may have zero or more outbound connection pools. Select which connection pool to configure a security credential map for. You can also choose to configure a credential map for the entire resource adapter. In this case, the credential map will apply to all connection pools within the resource adapter. Select Next.
8. A credential map can be configured for initial outbound connections, unauthentication connections, all WLS users or groups, or a particular WLS user or group. These correspond to the special names in WLS 8.1: wl-ra-initial, wl-ra-anonymous and wl-ra-default respectively. Select which of these you'd like to configure the credential map for. If you choose to configure a particular WLS user or group then enter the user or group in the text box that you defined in step 2. Select Next.
9. Click the name of the EIS user in the Remote User column of the Credential Maps table.
10. Enter the EIS user name and password in the corresponding text boxes.
11. Select Finish.

Defining Users and Groups

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

Defining Users

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

Defining users in a security realm involves specifying a unique name and password for each user that will access resources in the WebLogic Server security realm in the users window of the Administration Console. For more information, see *Managing WebLogic Security*.

Defining Groups

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

Security Policy Processing

Java Security may be used to protect WebLogic Resources. See "Using Java Security to Protect WebLogic Resources" for information on how to run WebLogic Server using Java Security. The J2CA 1.5 Specification defines default security policies for resource adapters running in an application server. It also defines a way for a resource adapter to provide its own specific security policies overriding the default. 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 has not defined specific security policies, 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.

Configuring Security Identities for Resource Adapters

The following sections discuss how you configure various security identities for WebLogic Server resoure adapters. For more information on setting identity properties, see "security" on page A-10.

Listing 8-2 illustrates how you would configure all 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</adapter-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 would 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</adapter-name>
<security>
```

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

default-principal-name: Default Identity

You can define a single security identity that will be used for 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, in order to make all calls from WebLogic Server into the resource adapter and manage all resource adapter management tasks, you can set up 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 Resource Adapter Management Tasks

You can define a management identity that will be 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 run resource adapter 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 run resource adapter 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. The 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 recommmends 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 Reauthentication Mechanisms

You specify authentication and reauthentication 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-mechansim" on page A-20 and "reauthentication-support" on page A-21.

CHAPTER 9

Packaging and Deploying Resource Adapters

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

WebLogic Server application deployment is covered in more detail in *Deploying WebLogic Server Applications*. This topics covered in this section discuss packaging and deployment procedures that are specific to resource adapters.

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 a common directory format. This same format is used when a resource adapter is packaged in an exploded directory format as a RAR. A resource adapter is structured as in the following example:

Listing 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 subdirectory called `META-INF`.
- An optional `MANIFEST.MF` also resides in `META-INF`. A manifest file is automatically generated by the JAR tool and is always the first entry in the JAR file. By default, it is named `META-INF/MANIFEST.MF`. The manifest file is the place where any meta-information about the archive is stored.
- 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`)
- The resource adapter can contain native libraries required by the resource adapter for interacting with the EIS. (For example, `windows.dll` and `unix.so`)
- The resource adapter can include documentation and related files not directly used by the resource adapter. (For example, `readme.html` and `/images/ra.jpg`)
- Ensure that any dependencies of a resource adapter on platform-specific native libraries are resolved.
- When a standalone resource adapter RAR is deployed, the resource adapter must be made available to all J2EE applications in the application server.
- When a resource adapter RAR packaged within a J2EE application EAR is deployed, the resource adapter must be made available only to the J2EE application with which it is packaged. (This specification-compliant behavior may be overriden 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).

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 WebLogic Server Applications" in *Developing WebLogic Server Applications*.

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 `weblogic-ra.xml` document type definition.
7. When the resource adapter classes and deployment descriptors are set up in the staging directory, you can create the RAR with a JAR command such as:

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

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

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

For more information on this topic, see "Creating and Modifying Resource Adapters: Main Steps" on page 3-2.

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

Deployment Descriptor

Also similar to Web Applications, EJBs, and Enterprise Applications, resource adapters use two deployment descriptors to define their operational parameters. The deployment descriptor `ra.xml` is the J2EE standard deployment descriptor provided by Sun Microsystems and the J2CA 1.5 Specification. The `weblogic-ra.xml` deployment descriptor is specific to WebLogic Server and defines operational parameters unique to WebLogic Server. See Appendix A, "weblogic-ra.xml Schema."

You can modify deployment descriptors using the following tools:

- Using the Administration Console Configuration tabs to view, modify, and (when necessary) persist deployment descriptor elements. Some of the descriptor element changes take place dynamically at runtime without requiring the resource adapter to be redeployed. Other descriptor elements will require the resource adapter to be redeployed.
- An XML Editor with scema validation, such as BEA XML Editor on dev2dev or XMLSpy. (An evaluation copy of XMLSpy is bundled with this version of WebLogic Server.) See *BEA dev2dev Online* at http://dev2dev.bea.com/index.jsp.

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

A new production redeployment feature enables you to 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 has start() and stop() methods defined on it. This type of resource adapter is not eligible for production redeployment. Resource adapters connect to one or more EISs 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. WebLogic Server now provides the capability to allow resource adapters to participate in 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 resoure adapter must be based on the J2CA 1.5 Specification. (Support for 1.0 resource adapters is not available.)
- The resource adapter must implement the Suspendable interface (see Listing 4-3, “Suspendable Interface,” on page 4-8).
- The resource adapter must be packaged inside an enterprise application (EAR file). Standalone production redeployment is not supported.
- The Suspendable.supportsVersioning() must return true when invoked by WebLogic Server.
- The `enable-access-outside-app` element must be set to false (in the weblogic-ra.xml descriptor).

The Process

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

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 4-12.
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 and/or outbound communications from the old to the new resource adapter. How this is done is up to the individual resource adapter developer.

WebLogic Server Deployment Plans

A WebLogic Server deployment plan is an optional XML file that configures an application for deployment to WebLogic Server. A deployment plan works by setting property values that would normally be defined in the WebLogic Server deployment descriptors, or by overriding property values already defined in a WebLogic Server deployment descriptor.

Deployment plans help you modify an application's WebLogic Server configuration for deployment into to multiple, differing WebLogic Server environments *without* modifying the packaged WebLogic Server deployment descriptor files. Configuration changes are applied by adding or changing variables in the deployment plan, which define both the location of the WebLogic Server descriptor properties to change and the value to assign to those properties.

For more information on deployment plans, see "WebLogic Server Deployment Plans."

APPENDIX A

weblogic-ra.xml Schema

This document provides a complete reference for the schema for the WebLogic Server-specific deployment descriptor `weblogic-ra.xml`. 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.

The following sections describe the complex deployment descriptor elements that can be defined in the `weblogic-ra.xml` deployment descriptor.

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.

Element	Required Optional	Description
`native-libdir`	Optional but 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 will *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 still set to `true`.

Element	Required Optional	Description
`enable-global-access-to-classes`	Optional	When set to true (default), the resource adapter will allow global access to its classes.
`work-manager`	Optional	This 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 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.

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 you can specify. 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 specify admin object instances using the `admin-object-instance` element. These correspond 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. See "admin-object-instance" on page A-16. For `admin-objects` subelements, see "admin-objects" on page A-14.

Element	Required Optional	Description
`outbound-resource-adapter`	Optional	This complex element is used to describe the outbound components of a resource adapter. There are three levels 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-29. 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-30. 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 `work-manager`.

Element	Required Optional	Description
`name`	Required	Specifies the name of the Work Manager. The Connector 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.

Element	Required Optional	Description
`response-time-request-class / fair-share-request-class / context-request-class / request-class-name`	Optional	You can choose between 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.

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.

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 8-7.

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-11 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.

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-12 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.

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.

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.

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.

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.

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.

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.

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. 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

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

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.
`connection-properties`	Optional	Defines all the properties that apply to the admin object 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.

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.

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-29.

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.

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-26.

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-mechansim`	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.

Element	Required Optional	Description
`reauthentication-support`	Optional	Specifies whether a particular connection factory supports re-authentication of existing Managed-Connection 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. This element must be one of the following: <reauthentication-support>true</reauthentication-support> <reauthentication-support>false</reauthentication-support>
`properties`	Optional	You can define the following element within `properties`: property here you define two more subelements: name and value
`res-auth`	Optional	Specifies whether to use container- or application-managed security. The values for this element are `Application` or `Container`. The default, if not specified, 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.

Element	Required Optional	Description
`initial-capacity`	Optional	Specifies the initial number of ManagedConnections, which WebLogic Server attempts to create during deployment. Failure to specify this value will result in WebLogic Server using its defined default value. 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. Failure to specify this value will result in WebLogic Server using its defined default value. 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. Failure to specify this value will result in WebLogic Server using its defined default value. Default Value: 1

Element	Required Optional	Description
shrinking-enabled	Optional	Specifies whether or not unused ManagedConnections will be destroyed and removed from the connection pool as a means to control system resources. Failure to specify this value will result in Weblogic using its defined default value. Value Range: true | false Default Value: true
shrink-frequency-seconds	Optional	Specifies the amount of time (in seconds) the Connection Pool Management will wait between attempts to destroy unused ManagedConnections. Failure to specify this value will result in Weblogic using its defined default value. Default Value: 900 seconds
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 likes the backend system being unavailable, this specified value could be exceeded due to factors outside the pools control. Default Value: 0
connection-creation-retry-frequency-seconds	Optional	The periodicity of retry attempts by the pool to create connections. Default Value: 0

Element	Required Optional	Description
`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	Sets the periodicity at which connections in the pool are tested. Default Value: 0
`test-connections-on-create`	Optional	Enables the testing of newly created connections. Value Range: true|false Default Value: false
`test-connections-on-release`	Optional	Enables testing of connections when they are being released back into the pool. Value Range: true|false Default Value: false
`test-connections-on-reserve`	Optional	Enables testing of connections when they are being reserved. Value Range: true|false Default Value: false
`profile-harvest-frequency-seconds`	Optional	Specifies how frequently the profile for the connection pool is being harvested.

Element	Required Optional	Description
`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 or not the resource adapter supports the ManagedConnectionFactory.matchManagedConnections() method. If the resource adapter does not support this method (always returns null for this method), then WebLogic Server bypasses this method call during a connection request. Value Range: true|false Default Value: true

logging

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

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. Failure to specify this value will result in WebLogic Server using its defined default value. Value Range: true | false Default Value: false

Element	Required Optional	Description
`rotation-type`	Optional	Sets the file rotation type. 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 when the size is unwieldy. Default Value: bySize
`number-of-files-limited`	Optional	Specifies whether the number of files that this server instance creates to store old messages should be limited. (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. Value Range: true | false Default Value: false

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. Value Range: true | false Default Value: true
`log-file-rotation-dir`	Optional	Specifies the directory path where the rotated log files will be stored.

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.

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-31.

connection-instance

You can define the following subelements under `connection-instance`.

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.

APPENDIX B

Resource Adapter Best Practices

This appendix provides some best practices for resource adapter developers.

Classloading Optimizations for Resource Adapters

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

If there are very few JARs in the RAR and if the JARs are relatively small in size, there will be no significant perceivable 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 perform one of the following recommended actions to avoid the problem:

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. This also eliminates the nesting of JARs and thus improves the performance of classloading significantly.

Connection Optimizations

BEA recommends that resource adapters implement the new 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 of 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 the J2CA 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. See section 2.2.2 “Common Client Interface” of the J2CA 1.5 Specification.

As a best practice, you should not store the InteractionSpec that the CCI resource adapter is required to implement in the resource adapter archive (RAR) JAR file. Instead, you should package it in a separate JAR file outside of the RAR 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 (such as is the case with the 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.