August 2014
Documentation for resource adapter users, deployers, and software developers that describes how to develop applications that include Java EE resource adapters to be deployed to WebLogic Server.
## Contents

### Preface

- Documentation Accessibility ................................................................. ix
- Conventions ................................................................................................ ix

### 1 Introduction and Roadmap

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

### 2 Understanding Resource Adapters

2.1 Overview of Resource Adapters .............................................................. 2-1
2.1.1 Comparing WebLogic Server and WebLogic Integration Resource Adapters 2-1
2.1.2 Inbound, Outbound, and Bidirectional Resource Adapters .................. 2-2
2.1.3 Comparing 1.0 Resource Adapters to 1.5 and 1.6 ................................. 2-2
2.1.4 Connector Architecture 1.6 Support ..................................................... 2-3
2.1.5 Additional Support Provided by the WebLogic Server Connector Container 2-4
2.2 Java EE Connector Architecture ............................................................. 2-5
2.2.1 Java EE Architecture Diagram and Components ................................ 2-5
2.2.2 System-Level Contracts .................................................................... 2-7
2.3 Resource Adapter Deployment Descriptors ............................................ 2-8

### 3 Creating and Configuring Resource Adapters

3.1 Creating and Configuring Resource Adapters: Main Steps ....................... 3-1
3.2 Modifying an Existing Resource Adapter ................................................. 3-3
3.3 Configuring the ra.xml File ..................................................................... 3-4
3.3.1 Creating the ra.xml File Manually ....................................................... 3-4
3.3.2 Using Metadata Annotations to Specify Deployment Information .......... 3-4
3.3.3 Resource Adapter XML Schema Definitions ....................................... 3-5
3.4 Configuring the weblogic-ra.xml File ..................................................... 3-5
3.4.1 Editing Resource Adapter Deployment Descriptors ............................... 3-6
3.4.1.1 Editing Considerations ............................................................... 3-6
3.4.1.2 Schema Header Information ...................................................... 3-6
3.4.1.3 Conforming Deployment Descriptor Files to Schema ...................... 3-7
4 Programming Tasks

4.1 Required Classes for Resource Adapters ....................................................... 4-1
4.2 Generic Work Context, ..................................................................................... 4-2
  4.2.1 Interfaces, Classes, and Methods Added to Support the Generic Work Context ... 4-2
  4.2.2 Deployment Descriptor Element Added to Support the Generic Work Context ..... 4-3
4.3 Programming a Resource Adapter to Perform as a Startup Class.................... 4-3
  4.3.1 Minimum Content of a Resource Adapter.................................................. 4-3
  4.3.2 Submitting a Work Instance.......................................................................... 4-4
  4.3.3 Retrying a Work Submission .......................................................................... 4-6
4.4 Suspending and Resuming Resource Adapter Activity........................................ 4-6
4.5 Extended BootstrapContext................................................................................ 4-9
  4.5.1 Diagnostic Context ID.................................................................................... 4-10
  4.5.2 Dye Bits ......................................................................................................... 4-10
  4.5.3 Callback Capabilities....................................................................................... 4-10
  4.5.4 Bean Validation............................................................................................. 4-10
  4.5.5 BeanManager............................................................................................... 4-11
4.6 Administered Object Uniqueness...................................................................... 4-11

5 Using Contexts and Dependency Injection in Resource Adapters

5.1 Overview .......................................................................................................... 5-1
5.2 Resource Adapter Bean Discovery..................................................................... 5-2
5.3 Obtaining Contextual References to Resource Adapter Beans.......................... 5-2
5.4 Invoking Resource Adapter Beans From Other Application Types...................... 5-2
5.5 Using Resource Adapters Deployed as CDI Bean Archives................................. 5-2
  5.5.1 BeanManager Support.................................................................................... 5-3
  5.5.2 Injection Points............................................................................................. 5-3
5.6 Using CDI with Resource Adapter Component Beans........................................ 5-4
  5.6.1 Resource Adapter Component Beans Must Not Be Managed Beans............... 5-5
5.6.2 Using Dependency Injection.......................................................................... 5-6
  5.6.2.1 Notes on Injection Usage........................................................................... 5-6
  5.6.2.2 Example.................................................................................................... 5-7

6 Connection Management

6.1 Connection Management Contract ................................................................. 6-1
  6.1.1 Connection Factory and Connection............................................................ 6-2
A weblogic-ra.xml Schema
A.1 weblogic-connector................................................................. A-1
A.2 work-manager ........................................................................ A-4
A.3 connector-work-manager..................................................... A-5
A.4 security .................................................................................. A-6
A.4.1 default-principal-name ....................................................... A-7
A.4.2 manage-as-principal-name ............................................... A-7
A.4.3 run-as-principal-name ...................................................... A-8
A.4.4 run-work-as-principal-name................................................ A-8
A.4.5 security-work-context ....................................................... A-8
A.4.5.1 caller-principal-default-mapped .................................. A-9
A.4.5.2 caller-principal-mapping ............................................. A-10
A.4.5.3 group-principal-mapping ............................................ A-10
A.5 properties ............................................................................. A-10
A.6 admin-objects......................................................................... A-10
A.6.1 admin-object-group............................................................ A-11
A.6.1.1 admin-object-instance ................................................ A-12
A.7 outbound-resource-adapter ................................................ A-12
A.7.1 default-connection-properties ........................................ A-13
A.7.1.1 pool-params ............................................................... A-14
A.7.1.2 logging .......................................................................... A-16
A.7.2 connection-definition-group ........................................... A-18
A.7.2.1 connection-instance .................................................... A-18

B Resource Adapter Best Practices
B.1 Classloading Optimizations for Resource Adapters ................ B-1
B.2 Connection Optimizations .................................................. B-1
B.3 Thread Management ............................................................ B-2
B.4 InteractionSpec Interface ..................................................... B-2
B.5 Using javax.jms.ConnectionFactory ................................. B-2
This preface describes the document accessibility features and conventions used in this guide—Programming Resource Adapters for Oracle WebLogic Server.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldface</td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td>italic</td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
This document describes how to develop applications that include Java EE resource adapters and how to deploy them on WebLogic Server. It is written for resource adapter users, deployers, and software developers, and 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 following sections describe the contents and organization of this guide—Programming Resource Adapters for Oracle WebLogic Server.

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

1.1 Document Scope and Audience

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 Section 1.3, "Related Documentation.”

It is assumed that the reader is familiar with Java EE and resource adapter concepts. The foundation document for resource adapter development is the JSR 322: Java EE Connector Architecture 1.6 (Final Release). See [http://jcp.org/aboutJava/communityprocess/final/jsr322/index.htm](http://jcp.org/aboutJava/communityprocess/final/jsr322/index.htm). Resource adapter developers should become familiar with the Java EE Connector Architecture 1.6 specification. This document, Programming Resource Adapters for Oracle WebLogic Server, 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.

1.2 Guide to This Document

This section, Chapter 1, "Introduction and Roadmap,” introduces the organization of this guide.
Related Documentation

- Chapter 2, "Understanding Resource Adapters," introduces you to the Oracle WebLogic Server implementation of the Java EE Connector Architecture as well as the resource adapter types and XML schema.
- Chapter 3, "Creating and Configuring Resource Adapters," describes how to create resource adapters using the Oracle WebLogic Server implementation of the Java EE Connector Architecture.
- Chapter 4, "Programming Tasks," describes programming tasks for resource adapters.
- Chapter 5, "Using Contexts and Dependency Injection in Resource Adapters," describes WebLogic Server support for Contexts and Dependency Injection (CDI) in resource adapter beans.
- Chapter 6, "Connection Management," introduces you to resource adapter connection management.
- Chapter 7, "Transaction Management," introduces you to the resource adapter transaction management.
- Chapter 8, "Message and Transactional Inflow," describes resource adapter messaging inflow and transactional inflow.
- Chapter 9, "Security," describes how to configure security for resource adapters.
- Chapter 10, "Packaging and Deploying Resource Adapters," discusses packaging and deploying requirements for resource adapters and provides instructions for performing these tasks.

1.3 Related Documentation

The foundation document for resource adapter development is JSR 322: Java EE Connector Architecture 1.6. Programming Resource Adapters for Oracle WebLogic Server document assumes you are familiar with the Java EE Connector Architecture specification, which contains design and development information that is specific to developing WebLogic Server resource adapters.

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

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

1.4 Examples for the Resource Adapter Developer

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

Two sets of resource adapter examples are provided with this release of WebLogic
Server. One set is compliant with the 1.5 Connector Architecture, and another set with
the 1.6 Connector Architecture (included among the Java EE 6 examples). Oracle
recommends that you examine, run, and understand these resource adapter examples
before developing your own resource adapters.

1.5 New and Changed Features in This Release

For a comprehensive listing of the new WebLogic Server features introduced in this
release, see What’s New in Oracle WebLogic Server.
New and Changed Features in This Release
This chapter describes WebLogic Server resource adapters, which provide connectivity to Enterprise Information Systems (EISes), and the Java EE Connector Architecture on which the resource adapters are based. This chapter also describes the deployment descriptors that define the structure and run-time behavior of a resource adapter that is deployed on WebLogic Server.

This chapter includes the following sections:

- Overview of Resource Adapters
- Java EE Connector Architecture
- Resource Adapter Deployment Descriptors

2.1 Overview of Resource Adapters

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

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

2.1.1 Comparing WebLogic Server and WebLogic Integration Resource Adapters

It is important to note the difference between WebLogic Integration (WLI) resource adapters and WebLogic Server resource adapters. WebLogic Integration resource adapters are written to be specific to WebLogic Server and, in general, are not deployable to other application servers. However, WebLogic Server resource adapters written without WLI extensions are deployable in any Java EE-compliant application server. This document discusses the design and implementation of non-WLI resource adapters.
2.1.2 Inbound, Outbound, and Bidirectional Resource Adapters

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 Java EE Connector Architecture 1.5 and 1.6 can be configured to have more than one simultaneous outbound connection. You can configure each outbound connection to have its own WebLogic Server-specific authentication and transaction support. See Section 6.2, "Configuring Outbound Connections."

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

- Inbound resource adapter (1.5 and 1.6 only) — Allows an EIS to call application components and perform work. All communication is initiated by the EIS. The resource adapter may request threads from WebLogic Server or create its own threads; however, this is not the Oracle-recommended approach. Oracle recommends that the resource adapter submit work by way of the WorkManager. See Chapter 8, "Message and Transactional Inflow."

- Bi-directional resource adapter (1.5 and 1.6 only) — Supports both outbound and inbound communication.

2.1.3 Comparing 1.0 Resource Adapters to 1.5 and 1.6

WebLogic Server supports resource adapters developed under versions 1.0, 1.5, and 1.6 of the Java EE Connector Architecture. Java EE Connector Architecture 1.0 restricts resource adapter communication to a single external system using one-way outbound communication. Java EE Connector Architecture 1.5 lifts this restriction. Other capabilities provided by and for 1.5 and 1.6 resource adapters that do not apply to 1.0 resource adapters include:

- Outbound communication from the application to multiple systems, such as Enterprise Information Systems (EISes) and databases. See Section 2.1.2, "Inbound, Outbound, and Bidirectional Resource Adapters."

- Inbound communication from one or more external systems such as an EIS to the resource adapter. See Section 8.2.1, "Handling Inbound Messages."

- Transactional inflow to enable a Java EE application server to participate in an XA transaction controlled by an external Transaction Manager by way of a resource adapter. See Section 8.4, "Transactional Inflow."

- A Work Manager provided by WebLogic Server to enable resource adapters to create threads through Work instances. See Section A.2, "work-manager."

- A life cycle contract for calling start() and stop() methods of the resource adapter by the application server. See Section 4.3, "Programming a Resource Adapter to Perform as a Startup Class."
Another important difference between 1.0 resource adapters and 1.5 and 1.6 resource adapters is regarding connection pools. For 1.5 and 1.6 resource adapters, you do not automatically get one connection pool per connection factory; you must configure a connection instance. You do so by setting the `connection-instance` element in the `weblogic-ra.xml` deployment descriptor.

Although WebLogic Server Full Platform is now compliant with JSR 322: Java EE Connector Architecture 1.6, it continues to fully support versions 1.0 and 1.5. In accordance with Connector Architecture 1.6, WebLogic Server supports schema-based deployment descriptors. Resource adapters that have been developed based on the Java EE Connector Architecture 1.0 use Document Type Definition (DTD)-based deployment descriptors. Resource adapters that are built on DTD-based deployment descriptors are still supported.

This document describes the development and use of 1.6 resource adapters.

### 2.1.4 Connector Architecture 1.6 Support

The major themes of Connector Architecture 1.6 that are supported in WebLogic Server Full Platform include the following:

- **Ease of development features**
  
  Connector Architecture 1.6 adds a number of features to simplify the development process, such as metadata annotations and support for "sparse" deployment descriptors. Metadata annotations can be embedded within resource adapter class files to specify deployment information, minimizing or even eliminating the need to manually create the `ra.xml` file. See Section 3.3.2, "Using Metadata Annotations to Specify Deployment Information."

- **Generic work context**
  
  A generic work context is the mechanism used by the resource adapter to propagate contextual information, such as the transaction context and security context, from the EIS to WebLogic Server during message delivery or submitting a work instance. For more information, see Section 4.2, "Generic Work Context."

- **Security context**
  
  Connector Architecture 1.6 defines a standard, generic security context that leverages the work done in JSR 196: Java Authentication Service Provider Interface for Containers. For more information, see Section 9.3, "Security Inflow."

- **Miscellaneous improvements, including:**
  
  - Integration of JSR 303: Bean Validation
  
  - Dynamic Reconfigurable Configuration Properties
    
    This includes the ability to designate specific properties of resource adapter component beans to be dynamically configurable, enabling those properties to be reconfigured at run time without requiring adapter restart or redeployment. See Section 3.4.2.1, "Dynamic Reconfigurable Configuration Properties."

  - The ability for a resource adapter to determine and classify the level of transaction support it can provide at run time. See Section 6.1.4, "Specifying and Obtaining Transaction Support Level."

  - Optional distributed Work processing, which gives an application server instance’s WorkManager the choice to distribute a Work instance submitted by
a resource adapter to another \texttt{WorkManager} residing in a different application server instance.

### 2.1.5 Additional Support Provided by the WebLogic Server Connector Container

WebLogic Server provides a number of features in its Connector container that supplement the JSR 322: Java EE Connector Architecture 1.6, including the following:

- Support for JSR 299: Contexts and Dependency Injection for the Java EE Platform (CDI) in embedded and global resource adapters. CDI defines a set of services for using injection to specify dependencies in an application. For more information, see Chapter 5, "Using Contexts and Dependency Injection in Resource Adapters."

- Additional runtime transaction level specification. WebLogic Server exposes information about the runtime transaction level in the \texttt{ConnectorConnectionPoolRuntimeMBean.RuntimeTransactionSupport} MBean attribute and in the Administration Console. For more information, see Section 7.1, "Supported Transaction Levels."

- Ability to lookup the \texttt{TransactionSynchronizationRegistry} object in JNDI, using the standard name of \texttt{java:comp/TransactionSynchronizationRegistry}. Oracle extends support by providing two additional global JNDI names: \texttt{javax/transaction/TransactionSynchronizationRegistry} and \texttt{weblogic/transaction/TransactionSynchronizationRegistry}. For more information, see \texttt{javax.transaction.TransactionSynchronizationRegistry}.

- Management and monitoring of long-running Work instances, including the number of current active work requests and the number of completed work requests, which WebLogic Server exposes on the \texttt{ConnectorWorkManagerRuntimeMBean} and in the Administration Console. See Section 3.6, "Long-Running Work Support."

- Additional support for the \texttt{javax.resource.spi.RetryableException} exception by extending it to outbound connection pools. When you try to get a connection from a suspended connection pool, WebLogic Server throws a \texttt{RetryableApplicationServerInternalException} that implements the \texttt{RetryableException} interface. You can then use the \texttt{RetryableException} instance to determine whether the failure is transient.

- Supplemental support for the security context in the Administration Console by providing a means to create inbound EIS-to-WebLogic principal mappings, which map EIS principals, such as users or groups defined in the EIS security domain, to corresponding principals in the WebLogic domain. For more information, see Section 9.3.1, "Inbound Principal Mappings."

- Support for module-level JSR 303: Bean Validation configuration. WebLogic Server extends Java EE 6 by supporting the optional use this bean configuration file to validate a resource adapter module.

- New methods on the \texttt{weblogic.connector.extensions.ExtendedBootstrapContext} that:
  - Provide a means for a resource adapter to look up the \texttt{Validator} and \texttt{ValidatorFactory} instances of its own beans for validation. See Section 4.5.4, "Bean Validation."
  - Return the resource adapter’s \texttt{BeanManager} instance to support CDI injection. See Section 4.5.5, "BeanManager," and Section 5.5, "Using Resource
Adapters Deployed as CDI Bean Archives.

- **Work Name Hint** — Names a *Work* instance and is used as part of the thread name assigned to a long-running *Work* instance. The `nameHint` forms part of the thread name and is used only for long-running work. For more information, see Section 3.6, "Long-Running Work Support."

## 2.2 Java EE Connector Architecture

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

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

For a more detailed overview of the connector architecture, see Section 3, "The Connector Architecture," in JSR 322: Java EE Connector Architecture 1.6.

See also "Resource Adapters and Contracts" in *The Java EE 6 Tutorial*.

### 2.2.1 Java EE Architecture Diagram and Components

*Figure 2–1* and the discussion that follows describe a WebLogic Server implementation of Connector Architecture 1.6.
The connector architecture shown in Figure 2–1 demonstrates a bi-directional resource adapter. The following components are used in outbound connection operations:

- A client application that connects to WebLogic Server, but also needs to interact with the EIS.
- An application component (an EJB or servlet) that the client application uses to submit outbound requests to the EIS through the resource adapter.
- The WebLogic Server Connector container in which the resource adapter is deployed. The container in this example holds the following:
  - A deployed resource adapter that provides bi-directional (inbound and outbound) communication to and from the EIS.
  - One or more connection pools maintained by the container for the management of outbound managed connections for a given ManagedConnectionFactory (in this case, MCF-2 - there may be more corresponding to different types of connections to a single EIS or even different EISes).
  - Multiple managed connections (MC1, MCn), which are objects representing the outbound physical connections from the resource adapter to the EIS.
  - Connection handles (C-handle) returned to the application component from the connection factory of the resource adapter and used by the application component for communicating with the EIS.

The following components are used for inbound connection operations:
One or more external message sources (MS1, MS2), which could be an Enterprise Information System (EIS) or Message Provider, and which send messages inbound to WebLogic Server.

One or more ActivationSpecs (Act Spec), each of which corresponds to a single MessageListener type (MLT-i).

A MessageEndpointFactory created by the EJB container and used by the resource adapter for inbound messaging to create proxies to MessageEndpoint instances (MDB instances from the MDB pool).

A message endpoint application (a message-driven bean (MDB) and possibly other Java EE components) that receives and handles inbound messages from the EIS through the resource adapter.

2.2.2 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 Java EE Connector Architecture. These contracts consist of SPI classes and interfaces that are required to be implemented in the application server and the EIS, so that the two systems can work cooperatively. The EIS side of these system-level contracts are implemented in the resource adapter’s Java classes. The following standard contracts are supported:

- Connection management contract — Enables WebLogic Server to pool connections to an underlying EIS and enables application components to connect to an EIS. Also allows efficient use of connection resources through resource sharing and provides controls for associating and disassociating connection handles with EIS connections.

- Transaction management contract — A contract between the transaction manager and an EIS that supports transactional access to EIS resource managers. Enables WebLogic Server to use its transaction manager to manage transactions across multiple resource managers.

- Transaction inflow contract — Allows a resource adapter to propagate an imported transaction to WebLogic Server. Allows a resource adapter to flow-in transaction completion and crash recovery calls initiated by an EIS. Transaction inflow involves the use of an external transaction manager to coordinate transactions.

- Security contract — Extends the connection management contract by providing secure access to an EIS and support for a secure application environment that reduces security threats to the EIS and protects valuable information resources managed by the EIS.

- Life cycle management contract — Enables WebLogic Server to manage the life cycle of a resource adapter. This allows bootstrapping a resource adapter instance during its deployment or application server startup, and notification to the resource adapter instance when it is undeployed or when the application server is being shut down.

- Work management contract — Allows a resource adapter to do work (monitor network endpoints, call application components, and so on) by submitting Work instances to WebLogic Server for execution.

- Generic work context contract — Enables a resource adapter to control the contexts in which the Work instances that it submits are executed by the WorkManager in WebLogic Server. A generic work context mechanism also
enables WebLogic Server to support new message inflow and delivery schemes, providing the resource adapter with a robust contextual Work execution environment that includes the ability to manage concurrent activity.

The generic work context contract standardizes the transaction context and the security context. JSR 322: Java EE Connector Architecture 1.6 defines this contract between the resource adapter and the application server in detail, including interfaces and classes, the thread model, rules for verifying and establishing contexts, error handling, event notifications, and so on.

- **Message inflow contract** — Allows a resource adapter to asynchronously or synchronously deliver messages to message endpoints residing in WebLogic Server independent of the specific messaging style, messaging semantics, and messaging infrastructure used to deliver messages. Also serves as the standard message provider pluggability contract that enables a wide range of message providers (Java Message Service, Java API for XML Messaging, and so on) to be plugged into WebLogic Server through a resource adapter.

These system-level contracts are described in detail in JSR 322: Java EE Connector Architecture 1.6.

### 2.3 Resource Adapter Deployment Descriptors

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

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

- **ra.xml** - The standard Java EE deployment descriptor. All resource adapters must be specified in an ra.xml deployment descriptor file. The schema for ra.xml is http://java.sun.com/xml/ns/javaee/connector_1_6.xsd.

  **Note:** Connector Architecture 1.6 introduces metadata annotations, which allow you to specify deployment information in resource adapter class files, thereby minimizing or eliminating the need to manually create the deployment descriptor file ra.xml.

- **weblogic-ra.xml** - This WebLogic Server-specific deployment descriptor contains elements related to WebLogic Server features such as transaction management, connection management, and security. This file is required for the resource adapter to be deployed to WebLogic Server. The schema for the weblogic-ra.xml deployment descriptor file is http://xmlns.oracle.com/weblogic/weblogic-connector/1.3/weblogic-connector.xsd. For a reference to the weblogic-ra.xml deployment descriptor, see Appendix A, "weblogic-ra.xml Schema."
This chapter describes how to create and configure a WebLogic Server resource adapter and also how to prepare the resource adapter for deployment.

This chapter includes the following sections:

- Creating and Configuring Resource Adapters: Main Steps
- Modifying an Existing Resource Adapter
- Configuring the ra.xml File
- Configuring the weblogic-ra.xml File
- Bean Validation Configuration File
- Long-Running Work Support
- Tooling Support

### 3.1 Creating and Configuring Resource Adapters: Main Steps

This section describes how to create a new WebLogic resource adapter. The next section, Section 3.2, "Modifying an Existing Resource Adapter," describes how to take an existing resource adapter and prepare it for deployment on WebLogic Server.

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

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

1. Write the Java code for the various classes required by resource adapter (ConnectionFactory, Connection, and so on) in accordance with JSR 322: Java EE Connector Architecture 1.6. These classes will be specified in the ra.xml file. For example:

```xml
<managedconnectionfactory-class>
com.sun.connector.blackbox.LocalTxManagedConnectionFactory
</managedconnectionfactory-class>

<connectionfactory-interface>
javax.sql.DataSource
</connectionfactory-interface>

<connectionfactory-impl-class>
javax.sql.Connection
</connectionfactory-impl-class>
```
For 1.6 adapters, you can embed metadata annotations in the resource adapter class files to specify deployment information, eliminating the need to create the ra.xml file manually. For more information, see Section 3.3, "Configuring the ra.xml File."

---

**Note:** The WebLogic Server implementation of Connector Architecture 1.6 includes support for Contexts and Dependency Injection. This support has implications on the set of annotations that may be used in resource adapter component beans, which are beans that define special components managed by the Connector container and that have a special life cycle. For more information, see Chapter 4, "Programming Tasks."

---

For more information about programming resource adapters, see Chapter 4, "Programming Tasks."

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

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

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

   **Note:** Java EE Connector Architecture 1.6 no longer requires the ra.xml file to be created manually. Instead, deployment information can be specified in metadata annotations. See Section 3.3, "Configuring the ra.xml File."

---

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

   Create a staging directory anywhere on your hard drive. Place the JAR file in the staging directory and the deployment descriptors in a subdirectory called META-INF.

   Then create the resource adapter archive by executing a jar command similar to the following in the staging directory:
jar cvf myRAR.rar *

Optionally, you can include the Bean Validation configuration file, META-INF/validation.xml, inside the JAR file. WebLogic Server uses the Bean Validation configuration file to validate the resource adapter module.

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

During testing, you may need to edit the resource adapter deployment descriptors. You can do this using the WebLogic Server Administration Console or manually using an XML editor or a text editor. For more information about editing deployment descriptors, see Section 3.4, "Configuring the weblogic-ra.xml File," and "Configure resource adapter properties" in the Oracle WebLogic Server Administration Console Online Help. See also Appendix A, "weblogic-ra.xml Schema," for detailed information on the elements in the deployment descriptor.

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

For information about these steps, see Chapter 10, "Packaging and Deploying Resource Adapters." See also Deploying Applications to Oracle WebLogic Server for detailed information about deploying components and applications in general.

3.2 Modifying an Existing Resource Adapter

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

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

2. Extract the contents of the resource adapter archive:
   cd c:/stagedir
   jar xf blackbox-notx.rar

   The staging directory should now contain the following:
   - A JAR file containing Java classes that implement the resource adapter
   - A META-INF directory containing the Manifest.mf and ra.xml files

   Execute these commands to see these files:
   c:/stagedir> ls
   blackbox-notx.rar
   META-INF
   c:/stagedir> ls META-INF
   Manifest.mf
   ra.xml

3. Create the weblogic-ra.xml file. This file is the WebLogic-specific deployment descriptor for resource adapters. In this file, you specify parameters for connection factories, connection pools, and security settings.
For more information, see Section 3.4, "Configuring the weblogic-ra.xml File," and also refer to Appendix A, "weblogic-ra.xml Schema," for information on the XML schema that applies to weblogic-ra.xml.

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

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

c:/stagedir> 1s META-INF
   Manifest.mf
   ra.xml
   weblogic-ra.xml

5. Create the resource adapter archive:

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

6. Deploy the resource adapter to WebLogic Server. For more information about packaging and deploying the resource adapter, see Chapter 10, "Packaging and Deploying Resource Adapters," and Deploying Applications to Oracle WebLogic Server.

### 3.3 Configuring the ra.xml File

All resource adapters must be specified in an ra.xml deployment descriptor file. For 1.0 or 1.5 resource adapters, you must create this file manually. If you are creating a 1.6 resource adapter, you can optionally specify metadata annotations in the resource adapter classes, eliminating the need to create the ra.xml file manually. The following sections explain how to configure the ra.xml file:

- Creating the ra.xml File Manually
- Using Metadata Annotations to Specify Deployment Information
- Resource Adapter XML Schema Definitions

For more information about creating a ra.xml file, you can also refer to JSR 322: Java EE Connector Architecture 1.6.

### 3.3.1 Creating the ra.xml File Manually

If your resource adapter does not already contain a ra.xml file, and you are creating a 1.0 or 1.5 resource adapter, you must manually create or edit an existing one to set the necessary deployment properties for the resource adapter. You can use a text editor or XML editor to edit the properties.

### 3.3.2 Using Metadata Annotations to Specify Deployment Information

The Java EE Connector Architecture 1.6 no longer requires you to manually create a ra.xml file. Instead, metadata annotations can be included in resource adapter classes to provide the same functions that are specified in the ra.xml file.

If you choose to specify all deployment information in a ra.xml file, the Java EE Connector Architecture 1.6 includes the metadata-complete element, which you include in the ra.xml file and set to true. Setting the metadata-complete element to true causes all metadata annotations included in the resource adapter classes to be ignored. If the metadata-complete element is not specified, or is set to false, WebLogic Server merges the information specified in the annotations with the
information specified in the ra.xml file at run time, and uses this merged information to deploy and manage the resource adapter.

For more information about deployment descriptors and annotations, see Chapter 18, "Metadata Annotations," of JSR 322: Java EE Connector Architecture 1.6. See also "Metadata Annotations" in The Java EE 6 Tutorial.

3.3.3 Resource Adapter XML Schema Definitions

The Java EE Connector Architecture 1.6 introduces changes to the ra.xml file schema, primarily to support ease-of-development features such as metadata annotations. For details about schema definition changes, see Section 20.7, "Resource Adapter XML Schema Definition," in JSR 322: Java EE Connector Architecture 1.6.

The schema for the ra.xml file for 1.0 and 1.5 resource adapters is http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd. For 1.6 adapters, the schema is http://java.sun.com/xml/ns/javaee/connector_1_6.xsd.

3.4 Configuring the weblogic-ra.xml File

In addition to supporting features of the standard resource adapter configuration ra.xml file, WebLogic Server defines an additional deployment descriptor file, the weblogic-ra.xml file. This file contains parameters that are specific to configuring and deploying resource adapters in WebLogic Server. This functionality is consistent with the equivalent weblogic-*.xml extensions for EJBs and Web applications in WebLogic Server, which also add WebLogic-specific deployment descriptors to the deployable archive. The basic RAR or deployment directory can be deployed to WebLogic Server without a weblogic-ra.xml file. If a resource adapter is deployed in WebLogic Server without a weblogic-ra.xml file, a template weblogic-ra.xml file populated with default element values is automatically added to the resource adapter archive. However, this automatically generated weblogic-ra.xml file is not persisted to the RAR; the RAR remains unchanged.

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

- Descriptive text about the connection factory.
- JNDI name bound to a connection factory. (Resource adapters developed based on JSR 322: Java EE Connector Architecture 1.6 are bound in the JNDI as objects independently of their ConnectionFactory objects.)
- Reference to a separately deployed connection factory that contains resource adapter components that can be shared with the current resource adapter.
- Connection pool parameters that set the following behavior:
  - Initial number of ManagedConnections that WebLogic Server attempts to allocate at deployment time.
  - Maximum number of ManagedConnections that WebLogic Server allows to be allocated at any one time.
  - Number of ManagedConnections that WebLogic Server attempts to allocate when filling a request for a new connection.
  - Whether WebLogic Server attempts to reclaim unused ManagedConnections to save system resources.
  - The time WebLogic Server waits between attempts to reclaim unused ManagedConnections.
3.4.1 Editing Resource Adapter Deployment Descriptors

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

3.4.1.1 Editing Considerations

To edit XML elements manually:

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

3.4.1.2 Schema Header Information

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

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

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

### 3.4.1.3 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_6.xsd** contains the schema for the standard ra.xml deployment file, required for all resource adapters. This schema is maintained as part of JSR 322: Java EE Connector Architecture 1.6 and is located at http://java.sun.com/xml/ns/javaee/connector_1_6.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://xmlns.oracle.com/weblogic/weblogic-connector/1.3/weblogic-connector.xsd.

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

### 3.4.2 Dynamic Descriptor Updates: Console Configuration Tabs

You can use the Administration Console to view, modify, and (when necessary) persist deployment descriptor elements. Some descriptor element changes take place dynamically at run time without requiring the resource adapter to be redeployed. Other descriptor elements require redeployment after changes are made. To use the Administration Console to configure a resource adapter, open Deployments and click the name of the deployed resource adapter. Use the Configuration tab to change the configuration of the resource adapter and the other tabs to control, test, or monitor the resource adapter.

For information about using the Administration Console, see "Configure resource adapter properties" in the Oracle WebLogic Server Administration Console Online Help.
3.4.2.1 Dynamic Reconfigurable Configuration Properties

Dynamic reconfigurable configuration properties are described in Section 5.3.7.6 of JSR 322: Java EE Connector Architecture 1.6. For 1.6 resource adapters, WebLogic Server supports dynamic reconfigurable configuration properties for the following adapter component beans:

- ResourceAdapter beans
- ManagedConnectionFactory beans
- Administered object beans

At run time, after you update the dynamically configurable properties on any of these adapter component beans, you must update the adapter to put changes into effect. Updating the adapter is a relatively lightweight operation during which WebLogic Server modifies the run-time bean instances without interfering with active connection pools or admin objects that do not have configuration updates. You do not need to update the adapter immediately. However, changes to properties on adapter component beans do not go into effect unless the beans are dynamically updated or the resource adapter is restarted.

The resource adapter should be designed carefully with regard to support for dynamic changes to its properties during run time. Depending on the services provided by the resource adapter, it might be critically important that some properties should never be reconfigured when the adapter is running; for example, the listen address and port number of a resource adapter used for the EIS connection (any reconfiguration of those properties should require the adapter to be shut down and restarted). WebLogic Server does not impose any requirements on an adapter component bean with regard to whether specific properties may or may not be designated as dynamically reconfigurable. It is entirely for the adapter developer to decide which adapter component beans support dynamic update and which do not.

3.4.2.2 Dynamic Configuration Parameters

For 1.6 adapters, WebLogic Server supports dynamic update on properties of Resource Adapter, ManagedConnectionFactory, and admin object beans. Using the Administration Console, you can modify the following configuration parameters on those beans dynamically, without requiring the resource adapter to be redeployed:

- Edit the adapter JNDI name
- Create and delete outbound connection pools
- Edit the connection pool JNDI name
- Create and delete admin objects
- Edit admin object JNDI names

3.4.2.3 Dynamic Pool Parameters

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

- initial-capacity
- max-capacity
- capacity-increment
- shrink-frequency-seconds
Configuring the weblogic-ra.xml File

- highest-num-waiters
- highest-num-unavailable
- connection-creation-retry-frequency-seconds
- connection-reserve-timeout-seconds
- test-frequency-seconds

3.4.2.4 Dynamic Logging Parameters

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

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

3.4.3 Automatic Generation of the weblogic-ra.xml File

A resource adapter archive (RAR) deployed on WebLogic Server must include a weblogic-ra.xml deployment descriptor file in addition to the ra.xml deployment descriptor file specified in JSR 322: Java EE Connector Architecture 1.6.

If a resource adapter is deployed in WebLogic Server without a weblogic-ra.xml file, a template weblogic-ra.xml file populated with default element values is automatically added to the resource adapter archive. However, this automatically generated weblogic-ra.xml file is not persisted to the RAR; the RAR remains unchanged. WebLogic Server instead generates internal data structures that correspond to default information in the weblogic-ra.xml file.

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

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

3.4.4 (Deprecated) Configuring the Link-Ref Mechanism

The Link-Ref mechanism was introduced in the 8.1 release of WebLogic Server to enable the deployment of a single base adapter whose code could be shared by multiple logical adapters with various configuration properties. For 1.5 resource adapters in the current release, the Link-Ref mechanism is deprecated and is replaced by the new Java EE libraries feature. However, the Link-Ref mechanism is still supported in this release for 1.0 resource adapters. For more information on Java EE libraries, see "Creating Shared Java EE Libraries and Optional Packages" in Developing Applications for Oracle WebLogic Server. To use the Link-Ref mechanism, use the ra-link-ref element in your resource adapter's weblogic-ra.xml file.
The deprecated and optional `ra-link-ref` element allows you to associate multiple deployed resource adapters with a single deployed resource adapter. In other words, it allows you to link (reuse) resources already configured in a base resource adapter to another resource adapter, modifying only a subset of attributes. The `ra-link-ref` element enables you to avoid - where possible - duplicating resources (such as classes, JARs, image files, and so on). Any values defined in the base resource adapter deployment are inherited by the linked resource adapter, unless otherwise specified in the `ra-link-ref` element.

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

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

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

### 3.5 Bean Validation Configuration File

In its support of JSR 303: Bean Validation, WebLogic Server extends Java EE 6 by providing a module-level bean validation configuration file. WebLogic Server supports the optional use this file to validate a resource adapter module.

The bean validation configuration file can be specified for a resource adapter module regardless of whether the resource adapter is deployed independently (as a standalone RAR) or as part of an enterprise application (EAR). If no bean validation configuration file is specified for an adapter module, WebLogic Server uses a default bean validation configuration to validate the resource adapter module.

The bean validation configuration file is named `validation.xml` and is included among the deployment descriptors in the `META-INF` subdirectory of the RAR.

For more information about bean validation, see Section 4.5.4, "Bean Validation."

### 3.6 Long-Running Work Support

Section 11.7 of JSR 322: Java EE Connector Architecture 1.6 defines two standard hints to control the quality-of-service (QoS) characteristics afforded to it by the `WorkManager`:

- **Work Name Hint** — Names a `Work` instance and is used as part of the thread name assigned to a long-running `Work` instance.
- **Long-running `Work` instance Hint** — Performs the same function as the WebLogic Server extension annotation `@LongRunning`, which allows you to schedule a `Work` instance in a separate thread and that also facilitates the control and monitoring capabilities of long-running `Work` instances.

WebLogic Server allows you to configure a limit on the number of long-running `Work` instances that can be submitted by a resource adapter to be executed.
concurrently. The default limit is 10. You can change the limit to higher value, but you need to exercise care not to overburden system resources.

This limit can be specified either by using the `max-concurrent-long-running-requests` element in the `weblogic-ra.xml` file or by setting `ConnectorWorkManagerRuntimeMBean.ActiveLongRunningRequests` attribute, which is exposed in the Administration Console. The `ConnectorWorkManagerRuntimeMBean` includes getter and setter methods on the `ActiveLongRunningRequests` and `CompletedLongRunningRequests` attributes that allow you to configure and monitor information about long-running work instances.

For more information, see Section 8.5, "Configuring and Managing Long-Running Work."

### 3.7 Tooling Support

WebLogic Server supports the following tools, which can be used to help with resource adapter development and deployment:

- **weblogic.appmerge**
  
  Performs validation checks metadata annotations. When used with the `-writeInferredDescriptors` option, `weblogic.appmerge` generates a merged `ra.xml` that combines deployment information specified in annotations with the contents of any pre-existing `ra.xml` file.

  __Note:__ After you run the `weblogic.appmerge` tool, make sure the `metadata-complete` element in the merged `ra.xml` is set to `true`. This prevents the deployer from processing annotations again, which improves overall deployment performance and reduces deployment time.

  For more information, see "Using weblogic.appmerge to Merge Libraries" in *Developing Applications for Oracle WebLogic Server*.

- **appc**
  
  Performs extensive validation checks on annotations, bean classes, `ra.xml`, `weblogic-ra.xml`, and the resource adapter deployment plan (`weblogic.appmerge` validates annotations only).

  The appc tool also:
  
  - Provides extensive reports that include both warnings and errors.
  - Is particularly useful for validating a resource adapter and ensuring that its configuration is correct without having to deploy it.

  For more information, see "appc Reference" in *Programming Enterprise JavaBeans for Oracle WebLogic Server*. 
This chapter describes the programming tasks for implementing a WebLogic Server resource adapter. It lists the Java classes that are required by the Java EE Connector Architecture; explains how to program a resource adapter to perform as a startup class; explains how to suspend and resume resource adapter activity; and describes the ExtendedBootstrapContext class.

This chapter includes the following sections:

- Required Classes for Resource Adapters
- Generic Work Context
- Programming a Resource Adapter to Perform as a Startup Class
- Suspending and Resuming Resource Adapter Activity
- Extended BootstrapContext

### 4.1 Required Classes for Resource Adapters

A resource adapter requires the following Java classes, in accordance with Connector Architecture 1.6:

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

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

```xml
<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>
```
In addition, if the resource adapter supports inbound messaging, the resource adapter will require an ActivationSpec class for each supported inbound message type. See Chapter 8, "Message and Transactional Inflow."

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

### 4.2 Generic Work Context

Connector Architecture 1.6 defines the generic work context, a mechanism for a resource adapter to propagate contextual information from an EIS to WebLogic Server during message delivery or when submitting a Work instance. The generic work context comprises a set of classes, interfaces, and methods, and also includes new schema elements supported in WebLogic Server.

The following sections describe these entities added to support the generic work context:

- Interfaces, Classes, and Methods Added to Support the Generic Work Context
- Deployment Descriptor Element Added to Support the Generic Work Context

#### 4.2.1 Interfaces, Classes, and Methods Added to Support the Generic Work Context

The following interfaces are added to support the generic work context:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.resource.spi.work.WorkContext</td>
<td>Serves as a standard mechanism for a resource adapter to propagate an imported context from an EIS to an application server.</td>
</tr>
<tr>
<td>javax.resource.spi.work.WorkContextLifeCycleListener</td>
<td>Models the various events that occur during the processing of the WorkContexts associated with a Work instance. This interface may be implemented by a WorkContext instance to receive notifications from the WorkManager when the WorkContext is set as the execution context of the Work instance it is associated with.</td>
</tr>
<tr>
<td>javax.resource.spi.work.WorkContextProvider</td>
<td>Specifies the methods a Work instance uses to associate a List of WorkContext instances to be set when the Work instance gets executed by a WorkManager.</td>
</tr>
</tbody>
</table>

The following class is added to support the generic work context:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.resource.spi.work.WorkContextErrorCodes</td>
<td>Models the possible error conditions that might occur during associating a WorkContext with a Work instance.</td>
</tr>
</tbody>
</table>
The following method is added to BootstrapContext interface to support the generic work context:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isContextSupported</td>
<td>A resource adapter can check an application server’s support for a particular WorkContext type through this method. This mechanism enables a resource adapter developer to dynamically change the WorkContexts submitted with a Work instance based on the support provided by the application server.</td>
</tr>
</tbody>
</table>

### 4.2.2 Deployment Descriptor Element Added to Support the Generic Work Context

To support the generic work context, the required-work-context element is added to the ra.xml file schema to represent a WorkContext class that is required by the resource adapter for WebLogic Server to support. For each WorkContext class that is required, an individual required-work-context element is specified.

Note that the @Connector metadata annotation can be used in a resource adapter source file to specify this deployment descriptor information. For more information, see Section 18.4, "@Connector," in JSR 322: Java EE Connector Architecture 1.6.

### 4.3 Programming a Resource Adapter to Perform as a Startup Class

The following sections describe programming a resource adapter to perform as a startup class:

- Minimum Content of a Resource Adapter
- Submitting a Work Instance
- Retrying a Work Submission

#### 4.3.1 Minimum Content of a Resource Adapter

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

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

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

Example 4–1 shows a resource adapter having a minimum resource adapter class. It is the absolute minimum resource adapter that you can develop (other than removing the println statements). In this example, the only work performed by the start() method is to print a message to stdout (standard out).
Example 4–1 Minimum Resource Adapter

```java
import javax.resource.spi.ResourceAdapter;
import javax.resource.spi.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();
    }
}
```

4.3.2 Submitting a Work Instance

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

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

Example 4–2 shows resource adapter that submits work instances to the Work Manager. The resource adapter starts some work in the start() method, thus serving as a Java EE-compliant startup class.

Example 4–2 Resource Adapter Using the Work Manager and Submitting Work Instances

```java
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 Java EE 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("ResourceAdapterWorker 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)
Suspending and Resuming Resource Adapter Activity

```
{  
   // 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;
}
```

4.3.3 Retrying a Work Submission

There are instances in which the submission of a `Work` instance by a resource adapter can experience a transient failure. For example, JSR 322: Java EE Connector Architecture 1.6 describes how you can use the optional `startTimeout` parameter in a `WorkManager` interface implementation to specify a time interval within which the execution of the `Work` instance must start. If a `Work` submission times out, a `WorkSubmissionFailure` occurs and a `WorkRejectedException` is generated.

JSR 322: Java EE Connector Architecture 1.6 states that the application server throws out a `RetryableWorkRejectedException` when it determines that the failure of a `Work` submission may due to transient causes. When it receives a `RetryableWorkRejectedException`, the resource adapter may retry submitting the `Work` instance. WebLogic Server supports the `RetryableWorkRejectedException` in the following transient failure situations:

- The `Work` instance was submitted to a suspended `Work Manager`.
- The `Work` submission has timed out.

**Note:** WebLogic Server extends retryable exception support to outbound connection pools if a connection instance attempts to connect to a suspended connection pool. For more information, see Section 6.2.2, "Retrying a Connection Attempt."

4.4 Suspending and Resuming Resource Adapter Activity

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

You then invoke the `resume()` method to signal that the inbound queue be drained and messages be delivered, or notify the EIS that message receipt was re-enabled. Basically, the `resume()` method allows the resource adapter to continue normal operations.
You initiate the `suspend()` and `resume()` methods by making a call on the resource adapter runtime MBeans programmatically, using WebLogic Scripting Tool, or from the WebLogic Server Administration Console. See "Start and stop a resource adapter" in the Oracle WebLogic Server Administration Console Online Help for more information.

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

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

- Suspend is called by the `suspend()` method on the connector component MBean.
- The production redeployment sequence of calls is invoked (when a new version of the application is deployed that contains the resource adapter). See Section 10.2.3.1, "Suspensible Interface and Production Redeployment."

Example 4–3 contains the `Suspensible` interface for resource adapters:

```
// Example 4–3 Suspensible 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;
```
Suspending and Resuming Resource Adapter Activity

/**
 * 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;

4.5 Extended BootstrapContext

If, when a resource adapter is deployed, it has a resource adapter JavaBean specified in
the resource-adapter-class element of its ra.xml descriptor, the WebLogic
Server connector container calls the start() method on the resource adapter bean as
required by JSR 322: Java EE Connector Architecture 1.6. 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 Connector Architecture 1.6.

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

- Diagnostic Context ID
- Dye Bits
- Callback Capabilities
- Bean Validation
- BeanManager
4.5.1 Diagnostic Context ID

In the WebLogic Server Diagnostic Framework, a thread may have an associated diagnostic context. A request on the thread carries its diagnostic context throughout its lifetime, as it proceeds along its path of execution. The ExtendedBootstrapContext allows the resource adapter developer to set a diagnostic context payload consisting of a String that can be used, for example, to trace the execution of a request from an EIS all the way to a message endpoint.

This capability can serve a variety of diagnostic purposes. For example, you can set the String to the client ID or session ID on an inbound message from an EIS. During message dispatch, various diagnostics can be gathered to show the request flow through the system. As you develop your resource adapter classes, you can make use of the setDiagnosticContextID() and getDiagnosticContextID() methods for this purpose.

Note the following regarding the contents of the diagnostic context payload:

- The payload can be viewed by other code in the same execution context, and it can also flow out of the process along with the Work instance. Therefore, you should ensure that the application does not include any sensitive data in the payload that, for example, could be returned by the getDiagnosticContextID() method.
- The payload can be overwritten by other code in the same execution context. Therefore, the application must never have a dependency on a specific context ID being available in the payload. In addition, the application should also verify that the context ID in the payload matches what is expected before using it.

For more information about the diagnostic context, see Configuring and Using the Diagnostics Framework for Oracle WebLogic Server.

4.5.2 Dye Bits

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

4.5.3 Callback Capabilities

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

4.5.4 Bean Validation

In its support of JSR 303: Bean Validation, WebLogic Server extends Java EE 6 by providing a module-level bean validation configuration file, which WebLogic Server uses to validate the resource adapter module.

There are circumstances in which you might want a resource adapter to perform validation on other bean instances that are managed by that resource adapter. Because a resource adapter does not have its own JNDI namespace, it cannot look up its own Validator and ValidatorFactory instances using JNDI. Instead, the resource adapter can inject those beans using CDI, or use the following methods on the ExtendedBootstrapContext interface to obtain instances of those beans:
- getValidator()
- getValidatorFactory()

### 4.5.5 BeanManager

To support JSR 299: Contexts and Dependency Injection for the Java EE Platform (CDI), WebLogic Server implements the `getBeanManager` method on the `ExtendedBootstrapContext` interface. A resource adapter can invoke this method to obtain its own `BeanManager` instance and perform CDI-style injection of managed beans inside the resource adapter.

For more information about using the `getBeanManager` method on the `ExtendedBootstrapContext` interface to use CDI, see Chapter 5, "Using Contexts and Dependency Injection in Resource Adapters."

### 4.6 Administered Object Uniqueness

Connector Architecture 1.6 allows a resource adapter to have multiple administered object classes that implement the same interface. However, there must be no more than one administered object definition with the same interface and class name combination (see Section 20.4.1, "Resource Adapter Provider" in JSR 322: Java EE Connector Architecture 1.6). The `adminobject-type-uniqueness` constraint has been added to the schema definition for the `ra.xml` file to define the `adminobject-interface` and `adminobject-class` combination.

In previous releases of WebLogic Server, the mapping of an admin object group defined in `weblogic-ra.xml` to the corresponding admin object defined in `ra.xml` was based on the admin object interface only. However, to support multiple admin object classes that have the same interface, WebLogic Server includes the optional `admin-object-class` sub-element of the `admin-object-group` element in `weblogic-ra.xml`. You can use the `admin-object-class` sub-element to define an admin object interface and class combination that WebLogic Server is able to map to the corresponding admin object defined in `ra.xml`.

When mapping an admin object group, WebLogic Server uses the following rules, which also ensure backward compatibility with 1.0 and 1.5 adapters:

- If the admin object group defined in `weblogic-ra.xml` includes both an admin object interface and class, WebLogic Server attempts to match that interface and class to the corresponding admin object definition in `ra.xml`.
- If the admin object group defined in `weblogic-ra.xml` includes only one admin object interface, and more than one matching admin object interface is defined in `ra.xml`, WebLogic Server generates an error.
- If the admin object group defined in `weblogic-ra.xml` includes only one admin object interface, and only one matching admin object interface is defined in `ra.xml`, that specific admin object interface is used.

Notes: Note the following restrictions:

- The use of a resource adapter’s `BeanManager` instance by a separate, caller thread is not supported.
- You cannot use a `BeanManager` instance to manage the life cycle of resource adapter component beans.

For more information about using the `getBeanManager` method on the `ExtendedBootstrapContext` interface to use CDI, see Chapter 5, "Using Contexts and Dependency Injection in Resource Adapters."
This chapter describes WebLogic Server support for JSR 299: Contexts and Dependency Injection for the Java EE Platform (CDI) in its implementation of Connector Architecture 1.6.

This chapter includes the following sections:

- Overview
- Resource Adapter Bean Discovery
- Obtaining Contextual References to Resource Adapter Beans
- Invoking Resource Adapter Beans From Other Application Types
- Using Resource Adapters Deployed as CDI Bean Archives
- Using CDI with Resource Adapter Component Beans

5.1 Overview

The CDI specification defines a set of services for using injection to specify dependencies in an application. CDI provides contextual life cycle management of beans, type-safe injection points, a loosely coupled event framework, loosely coupled interceptors and decorators, alternative implementations of beans, bean navigation through the Unified Expression Language (EL), and a service provider interface (SPI) that enables CDI extensions to support third-party frameworks or future Java EE components.

CDI support in the WebLogic Server implementation of Connector Architecture 1.6 is based on the following related specifications:


For additional general information about CDI, see:

- "Using Contexts and Dependency Injection for the Java EE Platform" in Developing Applications for Oracle WebLogic Server
- "Introduction to Contexts and Dependency Injection for the Java EE Platform" in the Java EE 6 Tutorial.
5.2 Resource Adapter Bean Discovery

If a resource adapter RAR is a bean archive, WebLogic Server treats all JAR files inside the RAR as bean archives that can be discovered, regardless of whether those JARs contain META-INF/beans.xml files. A resource adapter RAR is a bean archive if it has a bean archive descriptor file, beans.xml, in its META-INF directory.

WebLogic Server ignores beans.xml files that exist within nested JARs inside the RAR.

When an application is deployed as a resource adapter RAR bean archive, the WebLogic Server Connector container searches the following for beans and bean references:

- The resource adapter RAR
- All classes packaged directly inside the resource adapter RAR
- Every bean archive referenced by the adapter RAR

5.3 Obtaining Contextual References to Resource Adapter Beans

A resource adapter is different from a Web application or an EJB in that a resource adapter does not have its own JNDI namespace. That is, a resource adapter module does not have a java:comp, java:module, or java:app namespace. Therefore, it is not possible to bind a named managed bean to a resource adapter’s JNDI namespace, and it is also not possible to perform a lookup (as specified in the Java EE 6 Managed Beans Specification) from a resource adapter’s JNDI namespace or to use the Java EE 6 @Resource annotation to inject a predefined bean.

However, WebLogic Server provides the ExtendedBootstrapContext.getBeanManager() method. A resource adapter can invoke the getBeanManager method to expose the BeanManager instance of its adapter module.

5.4 Invoking Resource Adapter Beans From Other Application Types

The WebLogic Server Connector container does not support injecting CDI bean classes contained in a resource adapter RAR bean archive into other Web applications or EJBs. WebLogic Server support is limited to permitting CDI beans within an adapter RAR bean archive to be used or invoked by other caller Web applications or EJBs, provided that those CDI beans are not client proxies.

5.5 Using Resource Adapters Deployed as CDI Bean Archives

If the resource adapter is deployed as a CDI bean archive, the WebLogic Server Connector container provides the following support for CDI beans within the resource adapter itself:

- The ability to discover managed beans, decorators, interceptors, events, and so on, that are inside the deployed resource adapter
- Support for third-party portable extensions, as defined in Chapter 11, “Portable Extensions,” in JSR 299: Contexts and Dependency Injection for the Java EE Platform
- Support for the CDI features that are exposed by the BeanManager
Support for bean instantiation, injection, decorators, interceptors, events, and so on, for managed beans inside the resource adapter

Note the following:

- A resource adapter’s BeanManager instance is exposed by the getBeanManager method on the ExtendedBootstrapContext object.
- WebLogic Server supports the use of an adapter’s BeanManager only in the adapter’s own thread. An adapter’s BeanManager cannot be used in another application’s thread.
- The WebLogic Server Connector container supports the injection of built-in BeanManager bean types that are inside the resource adapter module; for example, injecting into the ResourceAdapter bean.
- The use of the Resource injection annotation on a resource adapter’s managed beans is not supported.

### 5.5.1 BeanManager Support

A resource adapter’s BeanManager can be used in either of the following situations:

- During the adapter deployment process, such as when the ResourceAdapter.start method is invoked
- Inside the Work.run method, which is scheduled by the resource adapter’s WorkManager instance

The WebLogic Server Connector container supports the injection of built-in BeanManager bean types in the resource adapter module. However, the use of a resource adapter’s BeanManager instance by a caller thread is not supported.

### 5.5.2 Injection Points

The WebLogic Server Connector container supports injection points for the following beans within a resource adapter deployed as a CDI bean archive:

- The following built-in beans, which JSR 299: Contexts and Dependency Injection for the Java EE Platform requires to be provided in a Java EE container:
  - UserTransaction — Provided by WebLogic JTA.
  - Principal — The caller principal set by the WebLogic Server Connector container. Its value is the current principal on the thread at the time this instance is used, not when it was injected.
  - ValidationFactory — The ValidationFactory instance of the resource adapter module itself and that is also accessible from the ExtendedBootstrapContext.getValidatorFactory method.
  - Validator — The Validator instance of the resource adapter module itself and that is also accessible from the ExtendedBootstrapContext.getValidator method.
- The BeanManager instance, as defined in Section 11.3 of JSR 299: Contexts and Dependency Injection for the Java EE Platform, of the resource adapter module itself that is accessible from the ExtendedBootstrapContext.getBeanManager method.
- Any managed bean that conforms to JSR 299: Contexts and Dependency Injection for the Java EE Platform and the Java EE 6 Managed Beans Specification, which is a part of JSR 316: Java Platform, Enterprise Edition 6 (Java EE 6) Specification.
Any special Connector Architecture 1.6 built-in beans of the following types that are part of the current resource adapter module:

- `javax.resource.spi.ResourceAdapter` allowing injection of a reference to the current resource adapter bean, which always refers to either: the `ResourceAdapter` bean instance of the current adapter module; or null if no `ResourceAdapter` bean is defined for the current resource adapter module.

- `javax.resource.spi.BootstrapContext` or `weblogic.connector.extensions.ExtendedBootstrapContext` allowing injection of a reference to either: the current resource adapter’s `BootstrapContext` bean instance; or null if no `ResourceAdapter` bean is defined for the current resource adapter module. This bean type is also available from a parameter in an invocation of the `ResourceAdapter.start(BootstrapContext ctx)` method.

- `javax.resource.spi.work.WorkManager` allowing injection of a reference to either: the current resource adapter’s `WorkManager` instance, which is available also from the `BootstrapContext.getWorkManager()` method; or null if no `ResourceAdapter` bean is defined for the current resource adapter module.

- `javax.resource.spi.XATerminator` allowing injection of a reference to either: the current resource adapter’s `XATerminator` instance, which is also available from the `BootstrapContext.getXATerminator()` method; or null if no `ResourceAdapter` bean is defined for the current resource adapter module.

- `javax.transaction.TransactionSynchronizationRegistry` allowing injection of a reference to the JTA `TransactionSynchronizationRegistry` instance, which is also available also from the `BootstrapContext.getTransactionSynchronizationRegistry()` method.

5.6 Using CDI with Resource Adapter Component Beans

WebLogic Server supports four types of beans called resource adapter component beans, which define special components managed by the WebLogic Server Connector container. Resource adapter component beans are "POJOs" (Plain Old Java Objects), but are created and managed by the resource adapter container and have a special lifecycle.

The adapter component bean types are:

- `ResourceAdapter` bean — Resource adapter class that implements `javax.resource.spi.ResourceAdapter` interface, which contains operations for life cycle management and message endpoint setup.

- `ManagedConnectionFactory` bean — JavaBean class that implements the `javax.resource.spi.ManagedConnectionFactory` interface and is a factory of both `ManagedConnection` and EIS-specific connection factory instances. This interface supports connection pooling by providing methods for matching and creation of a `ManagedConnection` instance.

- `ActivationSpec` bean — JavaBean class that implements the `javax.resource.spi.ActivationSpec` interface and that holds the activation configuration information for a message endpoint.
Administered objects, or admin objects — Optional set of JavaBean classes that represent objects specific to a messaging style or message provider.

The following metadata annotations may be used within resource adapter component beans:

- `@Connector`
- `@Activation`
- `@ConnectionDefinition`
- `@ConnectionDefinitions`
- `@AdministeredObject`

**Note:** The preceding annotations are new in Connector Architecture 1.6 and are recommended for use instead of the corresponding `ra.xml` elements.

The following sections include important information about the programming requirements for resource adapter component beans:

- Resource Adapter Component Beans Must Not Be Managed Beans
- Using Dependency Injection

For information about setting dynamically configurable properties on resource adapter component beans, see Section 3.4.2.1, "Dynamic Reconfigurable Configuration Properties."

### 5.6.1 Resource Adapter Component Beans Must Not Be Managed Beans

Resource adapter component beans must not be managed beans. However, the WebLogic Server Connector container does support CDI injection of managed beans, as defined in JSR 299: Contexts and Dependency Injection for the Java EE Platform, into a resource adapter component bean. WebLogic Server also supports the `PostConstruct` and `PreDestroy` annotations in adapter component beans as well.

**Notes:** Note the following:

- The WebLogic Server Connector container does not support managed beans that conform to the Java EE 6 Managed Beans Specification, which is a part of JSR 316: Java Platform, Enterprise Edition 6 (Java EE 6) Specification.
- For information about designing a managed bean that meets the conditions required by JSR 299, see "About Managed Beans" in *The Java EE 6 Tutorial*.

To ensure that a resource adapter component bean is not treated as a managed bean, WebLogic Server will fail to deploy the adapter if any of the following class-level annotations are used within an adapter component bean:

- The `javax.annotation.ManagedBean` annotation
- Any scope annotation
- Any qualifier annotation
5.6.2 Using Dependency Injection

In a resource adapter that is deployed as a CDI bean archive, the WebLogic Server Connector container supports CDI for adapter component beans once they are created and initialized.

To support Dependency Injection for resource adapter component beans, consistent with Section EE.5.20, "Support for Dependency Injection (JSR-330)" in the Java Platform, Enterprise Edition (Java EE) Specification, Version 6, the WebLogic Connector container does the following when initializing these beans:

1. Initializes the resource adapter component bean configuration properties using values in deployment descriptors.
2. Uses the `PostConstruct` annotation after dependency injection is done to perform any initialization.
3. Performs bean validation, consistent with JSR 303: Bean Validation, and for an `ActivationSpec` bean, invokes the `validate()` method.
4. For a `ResourceAdapter` bean, invokes the `start()` method.
5. Makes all resource adapter component beans available either by binding them to JNDI or exposing them to endpoint applications.

5.6.2.1 Notes on Injection Usage

Resource adapter component beans cannot be injected into other beans outside of the resource adapter module because they are not standard managed beans. That is, they are not visible outside the resource adapter module in a way that is consistent with JSR 299: Contexts and Dependency Injection for the Java EE Platform. You can design adapter component beans to support injection, but it is important to ensure that they are not treated like managed beans because the notion of request scope or session scope is meaningless in resource adapter component beans.

Injection is supported as follows:

- Field and method injection, but not constructor injection, is supported using the `javax.inject.Inject` annotation.
- Injected Fields, as defined in Section 3.8 of JSR 299: Contexts and Dependency Injection for the Java EE Platform, is supported.
- All injection points listed in Section 5.5.2, "Injection Points," are supported, such as `weblogic.transaction.UserTransaction` or `javax.resource.spi.BootstrapContext`.
- The `PostConstruct` and `PreDestroy` injection annotations are supported as follows:
For ResourceAdapter bean types, the @PostConstruct method is called after the configuration properties are initialized but before the start() method is called. In addition, the @PreDestroy method is called after the stop() method.

For other bean types, the @PostConstruct method is called after the configuration properties are initialized but before the bean is bound to JNDI. In addition, the @PreDestroy method is called when the resource adapter is undeployed or when the server is shut down.

For all beans, WebLogic Server performs bean validation consistent with its support for JSR 303: Bean Validation and also call the validate() method, if applicable, after calling the @PostConstruct method.

Events, as defined in Chapter 10, "Events," in JSR 299: Contexts and Dependency Injection for the Java EE Platform, are supported.

The Resource injection annotation is not supported in a resource adapter module.

### 5.6.2.2 Example

The following example shows that during resource adapter deployment, WebLogic Server first instantiates a MyResourceAdapter instance consistent with CDI. MyResourceAdapter is the ResourceAdapter component bean of the resource adapter module shown in this example because it is annotated with the Connector annotation. During deployment, WebLogic Server also:

- Instantiates MyBean and injects it into the MyResourceAdapter instance using the javax.inject.Inject annotation.
- Injects the Validator instance of this adapter module into the MyResourceAdapter instance.
- Injects the WorkManager and UserTransaction instances of this adapter module into MyBean.

```java
@Connector
public class MyResourceAdapter implements ResourceAdapter{
    private @Inject MyBean bean;
    private @Validator v;

    public void start(BootstrapContext ctx){
        v.validate(this, AnotherGroup.class);
        bean.do();
        ...
    }

    public class MyBean{
        private String name;
        private @WorkManager wm;
        private @UserTransaction ut;

        public String getName(){
            return name;
        }

        public void setName(String name) {
```
this.name = name;
}

public void do(){
     Work w = _
     wm.scheduleWork(w);
}
}
This chapter describes the connection management contract between WebLogic Server and a resource adapter, in accordance to the requirements in Connector Architecture 1.6. It also explains how to configure outbound and inbound connections and connection pool parameters, and how to test connections.

This chapter includes the following sections:

- Connection Management Contract
- Configuring Outbound Connections
- Configuring Inbound Connections
- Configuring Connection Pool Parameters
- Connection Proxy Wrapper - 1.0 Resource Adapters
- Reset a Connection Pool
- Testing Connections

For more information about the connection management contract, see Chapter 6, "Connection Management,” of JSR 322: Java EE Connector Architecture 1.6.

6.1 Connection Management Contract

One of the requirements of Connector Architecture 1.6 is the connection management contract. The connection management contract between WebLogic Server and a resource adapter:

- Provides a consistent application programming model for connection acquisition for both managed and non-managed (two-tier) applications.

- Enables a resource adapter to provide a connection factory and connection interfaces based on the common client interface (CCI) specific to the type of resource adapter and EIS. This enables JDBC drivers to be aligned with the Java EE Connector Architecture 1.6 with minimum impact on the existing JDBC APIs.

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

- Supports connection pooling.

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

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

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

An application looks up a connection factory instance in the Java Naming and Directory Interface (JNDI) namespace and uses it to obtain EIS connections. See Section 6.1.3, "Obtaining the ConnectionFactory (Client-JNDI Interaction)."

6.1.2 Resource Adapters Bound in JNDI Tree

Version 1.5 and 1.6 resource adapters can be bound in the JNDI tree as independent objects, making them available as system resources in their own right or as message sources for message-driven beans (MDBs). In contrast, version 1.0 resource adapters are identified by their ConnectionFactory objects bound in the JNDI tree.

In a version 1.5 or 1.6 resource adapter, at deployment time, the ResourceAdapter Bean (if it exists) is bound into the JNDI tree using the value of the jndi-name element, shown in the weblogic-ra.xml file. As a result, administrators can view resource adapters as single deployable entities, and they can interact with resource adapter capabilities publicly exposed by the resource adapter provider. For more information, see jndi-name in Appendix A, "weblogic-ra.xml Schema."

6.1.3 Obtaining the ConnectionFactory (Client-JNDI Interaction)

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

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

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

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

---

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

---

1. The application server uses a configured resource adapter to create physical connections to the underlying EIS.
2. The application component looks up a ConnectionFactory instance in the component’s environment by using the JNDI interface, as shown in Example 6–1.
Example 6–1  JNDI Lookup

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

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

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

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

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

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

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

    cx.close();

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

6.1.4 Specifying and Obtaining Transaction Support Level

Section 7.13 of JSR 322: Java EE Connector Architecture 1.6 specifies that a resource adapter may determine and classify the level of transaction support it can provide at runtime. To have the ability to specify the level of transaction support, a resource adapter’s ManagedConnectionFactory class must implement the TransactionSupport interface. If this interface is not implemented, the Connector container uses the transaction support specified in the merged result of the resource adapter’s ra.xml file and Connector annotations.

JSR 322: Java EE Connector Architecture 1.6 also defines the rules and priorities on the transaction support level determined from the ra.xml file, Connector annotation, and the TransactionSupport interface.

WebLogic Server supplements support for obtaining transaction support level by exposing the following two methods on the
    ConnectorConnectionPoolRuntimeMBean:

■ ConnectorConnectionPoolRuntimeMBean.getRuntimeTransactionSupport() — Return the real transaction support level in use for this Connector connection pool.

This value may also be viewed in the WebLogic Server Administration Console in the Resource Adapter: Monitoring: Outbound Connection Pools page.
6.1.5 Specifying an Unshareable ManagedConnectionFactory

In most cases, an adapter’s ManagedConnectionFactory supports connection sharing, as defined in section 7.9 of JSR 322: Java EE Connector Architecture 1.6. The specification also says that a connection can be made unshareable by setting res-sharing-scope to Unshareable in the caller application’s deployment descriptor or annotation.

However, it can be inconvenient to define an unshareable resource reference in the caller application. For example, the caller application may perform a look up to a ConnectionFactory pool from WebLogic’s global JNDI directly, but the application does not define unshareable resource references to this pool. WebLogic Server treats such use of the pools as shareable by default. As a result, if an adapter does not support connection sharing, the adapter will not work.

To circumvent this problem, WebLogic Server supports the public annotation weblogic.connector.extensions.Unshareable. This annotation can be used on a ManagedConnectionFactory class if the ManagedConnectionFactory does not support sharing. When such an adapter is deployed, WebLogic Server checks the ManagedConnectionFactory class and treats the ManagedConnectionFactory and related pools as unshareable. If you configure a sharable resource reference to this unshareable pool in a Web application or an Enterprise Java Bean, WebLogic Server issues a warning message—but the Web application or the EJB nevertheless treats the pool as unshareable. There is no need to configure anything in weblogic-ra.xml or in the Administration Console.

If a ManagedConnectionFactory is shareable, nothing needs to be changed in the adapter’s code. All ManagedConnectionFactory instances and pools are considered shareable by default, unless the ManagedConnectionFactory contains an Unshareable annotation.

6.2 Configuring Outbound Connections

Outbound resource adapters based on Connector Architecture 1.6 can be configured to have one or more outbound connections, each having its own WebLogic Server-specific authentication and transaction support. You configure outbound connection properties in the ra.xml and weblogic-ra.xml deployment descriptor files.

6.2.1 Connection Pool Configuration Levels

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

You can define outbound connection pools at three levels:

- Global - Specify parameters that apply to all outbound connection groups in the resource adapter using the default-connection-properties element. See Section A.7.1, "default-connection-properties."
- Group - Specify parameters that apply to all outbound connection instances belonging to a particular connection factory specified in the ra.xml deployment
Configuring Outbound Connections

6.1 Connection Management

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 Section A.7.2, "connection-definition-group."

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

- Instance - Under each connection definition group, you can specify connection instances using the connection-instance element of the weblogic-ra.xml deployment descriptor. These correspond to the individual connection pools for the resource adapter. You can use the connection-properties subelement to specify properties at the instance level too; properties specified at the instance level override those provided at the group and global levels. See Section A.7.2.1, "connection-instance."

6.2 Retrying a Connection Attempt

If an application component attempts to obtain a connection instance from a connection pool using the getConnection() method on the ConnectionFactory, but the pool is temporarily suspended, WebLogic Server generates an exception that implements javax.resource.spi.RetryableException. The application component can use an instance of RetryableException to determine whether the connection failure is transient.

6.3 Multiple Outbound Connections Example

Example 6–2 is an example of a weblogic-ra.xml deployment descriptor that configures multiple outbound connections:

Example 6–2  weblogic-ra.xml Deployment Descriptor: Multiple Outbound Connections

<?xml version='1.0'?>
<weblogic-connector xmlns="http://xmlns.oracle.com/weblogic/weblogic-connector">
  <jndi-name>900eisaNameOfBlackBoxXATx</jndi-name>
  <outbound-resource-adapter>
    <connection-definition-group>
      <connection-factory-interface>javax.sql.DataSource</connection-factory-interface>
      <connection-instance>
        <jndi-name>eis/900eisaBlackBoxXATxConnectorJNDINAME1</jndi-name>
        <connection-properties>
          <pool-params>
            <initial-capacity>2</initial-capacity>
            <max-capacity>10</max-capacity>
            <capacity-increment>1</capacity-increment>
            <shrinking-enabled>true</shrinking-enabled>
            <shrink-frequency-seconds>60</shrink-frequency-seconds>
          </pool-params>
          <properties>
            <property>
              <name>ConnectionURL</name>
              <value>
jdbc:oracle:thin:@bcpdb:1531:bay920;create=true;autocommit=false
</value>
</property>
</connection-definition-group>
<connection-definition-group>
<connection-factory-interface>javax.sql.DataSourceCopy</connection-factory-interface>
<connection-instance>
<jndi-name>eis/900eisaBlackBoxXATxConnectorJNDINAME3</jndi-name>
<connection-properties>
<pool-params>
<initial-capacity>2</initial-capacity>
<max-capacity>10</max-capacity>
<capacity-increment>1</capacity-increment>
<shrinking-enabled>true</shrinking-enabled>
<shrink-frequency-seconds>60</shrink-frequency-seconds>
</pool-params>
<properties>
<property>
<name>ConnectionURL</name>
</property>
</connection-properties>
</connection-instance>
</connection-definition-group>
<initial-capacity>2</initial-capacity>
<max-capacity>10</max-capacity>
<capacity-increment>1</capacity-increment>
<shrinking-enabled>true</shrinking-enabled>
<shrinking-frequency-seconds>60</shrinking-frequency-seconds>
</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>

6.3 Configuring Inbound Connections

Connector Architecture 1.6 permits you to configure a resource adapter to support
inbound message connections. The following are the main steps for configuring an
inbound connection:

1. Provide a JNDI name for the resource adapter in the weblogic-ra.xml
deployment descriptor. See jndi-name in Table A-1, "weblogic-connector
subelements"

2. Configure a message listener and ActivationSpec for each supported inbound
message type in the ra.xml deployment descriptor. For information about
requirements for an ActivationSpec class, see Chapter 13, "Message Inflow" in
JSR 322: Java EE Connector Architecture 1.6.

3. Within the packaged enterprise application, include a configured EJB
message-driven bean (MDB). In the resource-adapter-jndi-name element of
the weblogic-ejb-jar.xml deployment descriptor, provide the same JNDI
name assigned to the resource adapter in the previous step. Setting this value
enables the MDB and resource adapter to communicate with each other.

4. Configure the security identity to be used by the resource adapter for inbound
connections. When messages are received by the resource adapter, work must be
performed under a particular security identity. See Section 9.5, "Configuring
Security Identities for Resource Adapters."

5. Deploy the resource adapter as discussed in Deploying Applications to Oracle
WebLogic Server.
6. Deploy the MDB. For more information, see "Message-Driven EJBs" in Programming Enterprise JavaBeans for Oracle WebLogic Server and Deploying Applications to Oracle WebLogic Server.

Example 6–3 shows how an inbound connection with two message listener/activation specs could be configured in the ra.xml deployment descriptor:

**Example 6–3  Example of Configuring an Inbound Connection**

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

6.4 Configuring Connection Pool Parameters

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

6.4.1 initial-capacity: Setting the Initial Number of ManagedConnections

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

Because no initiating security principal or request context information is known at WebLogic Server startup, a server instance creates initial connections using a security subject by looking up special credential mappings for the initial connection. See Section 9.2.2.2, "Initial Connection: Requires a ManagedConnection from Adapter Without Application’s Request."
6.4.2 max-capacity: Setting the Maximum Number of ManagedConnections

As more ManagedConnections are created, they consume more system resources - such as memory and disk space. Depending on the Enterprise Information System (EIS), this consumption may affect the performance of the overall system. To control the effects of ManagedConnections on system resources, you can specify a maximum number of allocated ManagedConnections in the max-capacity element of the weblogic-ra.xml descriptor file.

If a new ManagedConnection (or more than one ManagedConnection in the case of capacity-increment being greater than one) needs to be created during a connection request, WebLogic Server ensures that no more than the maximum number of allowed ManagedConnections are created. Requests for newly allocated ManagedConnections beyond this limit results in a ResourceAllocationException being returned to the caller.

6.4.3 capacity-increment: Controlling the Number of ManagedConnections

In compliance with Connector Architecture 1.6, when an application component requests a connection to an EIS through the resource adapter, WebLogic Server first tries to match the type of connection being requested with an existing and available ManagedConnection in the connection pool. However, if a match is not found, a new ManagedConnection may be created to satisfy the connection request.

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

6.4.4 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.

6.4.5 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.

Note: WebLogic Server uses null as Subject if a mapping is not found.
6.4.6 highest-num-waiters: Controlling the Number of Clients Waiting for a Connection

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

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

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

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

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

6.4.9 match-connections-supported: Matching Connections

A connection request contains parameter information. By default, the connector container calls the `matchManagedConnections()` method on the `ManagedConnectionFactory` to match the available connection in the pool to the parameters in the request. The connection that is successfully matched is returned.

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

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

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

6.4.10 test-frequency-seconds: Testing the Viability of Connections

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

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

You can set the `test-connections-on-create` element to enable the testing of connections as they are created. The default value is false.
6.4.12 test-connections-on-release: Testing Connections upon Release to Connection Pool

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

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

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

6.5 Connection Proxy Wrapper - 1.0 Resource Adapters

The connection proxy wrapper feature is valid only for resource adapters that are created based on the Java EE Connector Architecture 1.0. 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

6.5.1 Possible ClassCastException

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

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

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

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

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

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

1. Rather than having `MyConnection` be a class that implements the resource adapter's `Connection` interface, modify `MyConnection` to be an interface that extends `Connection`.
2. Implement a `MyConnectionImpl` class that implements the `MyConnection` interface.
6.5.2 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.

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.

6.6 Reset a Connection Pool

You may need to reset a connection pool to:

- Recover a connection pool in an unhealthy state without interfering other running connection pools.
- Make non-dynamic configuration changes that could not take effect through an update operation. For example: changing properties on a `ManagedConnectionFactory` or changing transaction support for connection.

You can reset a connection pool in one of two ways:

- Reset—If no connections in the pool are in use, the pool is recreated. The new pool includes any configuration changes you may have made prior to the reset. If a connection is in use, the pool is not reset.
- Force Reset—Immediately discards all used and unused connections and the pool is recreated. The new pool includes any configuration changes you may have made prior to the reset.

Use the following steps to reset a connection pool from the administration console:

1. Select your resource adapter from the Summary of Deployments table.
2. Select Control > Outbound Connection Pools
3. Select the connection pools to reset.
4. Click Reset or Force Reset.

6.7 Testing Connections

If a resource adapter's `ManagedConnectionFactory` implements the `Validating` interface, then the application server can test the validity of existing connections. You
can test either a specific outbound connection or the entire pool of outbound connections for a particular ManagedConnectionFactory. Testing the entire pool results in testing each connection in the pool individually. For more information on this feature, see section 6.5.3.4 “Detecting Invalid Connections” in JSR 322: Java EE Connector Architecture 1.6.

6.7.1 Configuring Connection Testing

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

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

- **test-connections-on-create** - Determines whether the connection should be tested upon its creation. By default it is false.

- **test-connections-on-release** - Determines whether the connection should be tested upon its release. By default it is false.

- **test-connections-on-reserve** - Determines whether the connection should be tested upon its reservation. By default it is false.

6.7.2 Testing Connections in the Administration Console

To test a resource adapter’s connection pools:

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

2. Select the Test tab.

   You will see a table of connection pools for the resource adapter and the test status of each pool.

3. Select the connection pool you want to test and click Test.

See “Test outbound connections” in the Oracle WebLogic Server Administration Console Online Help.
This chapter describes the system-level transaction management contract that is used for outbound communication from WebLogic Server to Enterprise Information Systems.

This chapter includes the following sections:

- Supported Transaction Levels
- Configuring Transaction Levels

For more information about transaction management, see Chapter 7, "Transaction Management," in JSR 322: Java EE Connector Architecture 1.6. For information about transaction management for inbound communication from EISes to WebLogic Server, see Section 8.4, "Transactional Inflow."

### 7.1 Supported Transaction Levels

A transaction is a set of operations that must be committed together or not at all for the data to remain consistent and to maintain data integrity. Transactional access to EISes is an important requirement for business applications. The Java EE Connector Architecture 1.6 supports the use of transactions.

WebLogic Server utilizes the WebLogic Server Transaction Manager implementation and supports resource adapters having XA, local, or no transaction support. You define the type of transaction support in the transaction-support element in the ra.xml file; a resource adapter can support only one type. You can use the transaction-support element in the weblogic-ra.xml deployment descriptor to override the value specified in ra.xml. See Section 7.2, "Configuring Transaction Levels," and transaction-support in Table A–18, "default-connection-properties subelements" for details.

Resource adapters conforming to Java EE Connector Architecture 1.6 can optionally specify the level of transaction support at run time. This requires the implementation of the TransactionSupport interface. For more information, see Section 6.1.4, "Specifying and Obtaining Transaction Support Level."

#### 7.1.1 XA Transaction Support

XA transaction support allows a transaction to be managed by a transaction manager external to a resource adapter (and therefore external to an EIS). When an application component demarcates an EIS connection request as part of a transaction, the application server is responsible for enlisting the XA resource with the transaction manager. When the application component closes that connection, the application server cleans up the EIS connection once the transaction has completed.
Oracle recommends creating a LocalTransaction outbound connection pool for an XA transaction capable resource adapter for improved performance.

7.1.2 Local Transaction Support

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

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

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

7.1.3 No Transaction Support

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

7.1.4 Runtime Transaction Support Level Specification

JSR 322: Java EE Connector Architecture 1.6 states that a resource adapter may optionally determine and classify the level of transaction support it can provide at run time. To expose information about the level of transaction support at run time, a ManagedConnectionFactory must implement the TransactionSupport interface. JSR 322: Java EE Connector Architecture 1.6 also defines rules and priorities on transaction support levels set in descriptors, annotations, and the TransactionSupport interface. For example, WebLogic Server uses the value returned by the getTransactionSupport method and ignores the value specified by the resource adapter’s deployment descriptor and the @Connector metadata annotation.

WebLogic Server exposes information about the runtime transaction support level in the ConnectorConnectionPoolRuntimeMBean.RuntimeTransactionSupport MBean attribute and also in the Administration Console.

To view the runtime transaction level support in the Administration Console:

1. In the Summary of Deployments page, select the resource adapter.
2. Click Monitoring > Outbound Connection Pools, and view the items in the Runtime Transaction Support column.
### 7.2 Configuring Transaction Levels

You specify a transaction support level for a resource adapter in the Java EE 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:
  ```xml
  <transaction-support>NoTransaction</transaction-support>
  ```

- For XA Transaction, add the following entry to the *ra.xml* deployment descriptor file:
  ```xml
  <transaction-support>XATransaction</transaction-support>
  ```

- For Local Transaction, add the following entry to the *ra.xml* deployment descriptor file:
  ```xml
  <transaction-support>LocalTransaction</transaction-support>
  ```

Resource adapters conforming to Java EE Connector Architecture 1.6 can optionally specify the level of transaction support at run time. This requires the implementation of the `TransactionSupport` interface. For more information, see Section 6.1.4, "Specifying and Obtaining Transaction Support Level."

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

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

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

For more information on specifying the transaction level in the *ra.xml* deployment descriptor, see Section 20.7, "Resource Adapter XML Schema Definition," in JSR 322: Java EE Connector Architecture 1.6. For more information on specifying the transaction level in the `weblogic-ra.xml` deployment descriptor, see Appendix A, "weblogic-ra.xml Schema."

#### 7.2.1 Configure XA Transaction Recovery Credential Mapping

For pools which support XA Transactions, WebLogic Server may try to perform transaction recovery for the Java EE Connector Architecture connection pool if WebLogic Server finds pending transactions in the pool during a server startup. If pending transactions are found, WebLogic Server gets a ManagedConnection to EIS during recovery using

```java
ManagedConnectionFactory.createManagedConnection(javax.security.auth.Subject subject, ConnectionRequestInfo cxRequestInfo).
```

If EIS requires explicit credentials (such as user name and password) to sign-on, the you need to configure WebLogic Server with appropriate credentials by configuring a special credential mapping for the initial connection. See Section 9.2.2.2, "Initial Connection: Requires a ManagedConnection from Adapter Without Application's Request.” WebLogic Server uses `null` as `Subject` if a mapping is not found.

---

**Note:** You don’t need to configure this special credential mapping if the EIS doesn’t require explicit credentials.
This chapter describes how WebLogic resource adapters use inbound connections to handle message inflow and transactional inflow.

This chapter includes the following sections:

- Overview of Message and Transactional Inflow
- How Message Inflow Works
- Message Inflow to Message Endpoints (Message-driven Beans)
- Transactional Inflow
- Configuring and Managing Long-Running Work

### 8.1 Overview of Message and Transactional Inflow

Message inflow refers to inbound communication from an EIS to the application server, using a resource adapter. Inbound messages can be part of a transaction that is governed by a Transaction Manager that is external to WebLogic Server and the resource adapter, as described in Section 8.4, "Transactional Inflow."

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.
8.1.1 Architecture Components

Figure 8–1 contains the following components:

■ A client application, which connects to an application running on WebLogic Server, but which also needs to connect to an EIS

■ An external system (in this case, an EIS or Enterprise Information System)

■ An application component (an EJB) that the client application uses to submit outbound requests to the EIS through the resource adapter

■ A message endpoint application (a message-driven bean and possibly other Java EE components) used for the receipt of inbound messages from the EIS through the resource adapter

■ The WebLogic Server Work Manager and an associated thread (or threads) to which the resource adapter submits Work instances to process inbound messages and possibly process other actions.

■ An external Transaction Manager, to which the WebLogic Server Transaction Manager is subordinate for transactional inflow of messages from the EIS

■ The WebLogic Server Connector container in which the resource adapter is deployed. The container manages the following:
  - A deployed resource adapter that provides bi-directional (inbound and outbound) communication to and from the EIS.
  - An active Work instance.
Multiple managed connections (MC1, ..., MCn), which are objects representing the outbound physical connections from the resource adapter to the EIS.

Connection handles (C-handle) returned to the application component from the connection factory of the resource adapter and used by the application component for communicating with the EIS.

One of perhaps many activation specifications. There is an activation specification (ActivationSpec) that corresponds to each specific message listener type, MLT-j. For information about requirements for an ActivationSpec class, see Chapter 13, "Message Inflow" in JSR 322: Java EE Connector Architecture 1.6.

One of the connection pools maintained by the container for the management of managed connections for a given ManagedConnectionFactory (in this case, MCF-2. A Connector container could include multiple connection pools, each corresponding to a different type of connections to a single EIS or even different EISes).

A MessageEndpointFactory created by the EJB container and used by the resource adapter to create proxies to MessageEndpoint instances (MDB instances from the MDB pool).

An external message source, which could be an EIS or Message Provider

### 8.1.2 Inbound Communication Scenario

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

A typical simplified inbound sequence involves the following steps:

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

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

- A proprietary communications channel and protocol for connecting to and communicating with an EIS. The communications channel and protocol are not visible to the application server in which the resource adapter is deployed. See Appendix 8.2.2, "Proprietary Communications Channel and Protocol."
- 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.

8.2.1 Handling Inbound Messages

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

- Handle the message locally, that is, within the ResourceAdapter bean, without involving other components.
- Pass the message off to another application component. For example, it may look up an EJB and invoke a method on it.
- Send the message to a message endpoint. Typically, a message endpoint is a message-driven bean (MDB). For more information, see Appendix 8.3, "Message Inflow to Message Endpoints (Message-driven Beans)."

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

Depending upon the transactional capabilities of the resource adapter and the EIS, inbound messages can be either part of a transaction (XA) or not (non-transactional). If the messages are XA, the controlling transaction may be coordinated by an external Transaction Manager (transaction inflow) or by the application server's Transaction Manager. See Appendix 8.4, "Transactional Inflow."

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.

8.2.2 Proprietary Communications Channel and Protocol

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

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

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

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

For more information about message-driven Beans, see "Message-Driven EJBs" in *Programming Enterprise JavaBeans for Oracle WebLogic Server*.

8.3.1 Deployment-Time Binding Between an MDB and a Resource Adapter

A resource adapter can be deployed independently (as a standalone RAR) or as part of an enterprise application (EAR). An MDB can also be deployed independently (as a standalone JAR) or as part of an enterprise application (EAR). In either case, an MDB whose messages are derived from a resource adapter must be bound to the resource adapter. The following sections describe binding the MDB and resource adapter and subsequent messaging operations.

8.3.1.1 Binding an MDB and a Resource Adapter

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

1. Set the `jndi-name` element in the `weblogic-ra.xml` deployment descriptor for the resource adapter. See `jndi-name` in Appendix A, "weblogic-ra.xml Schema."

2. Set the `adapter-jndi-name` element in the `weblogic-ejb-jar.xml` deployment descriptor to match the value set in the corresponding `jndi-name` element in the resource adapter.

3. Assume that the resource adapter is deployed prior to the MDB. (The MDB could be deployed before the resource adapter is deployed; in that case, the deployed MDB polls until the resource adapter is deployed.) When the resource adapter is deployed, the `ResourceAdapter` bean is bound into JNDI using the name specified.

4. The MDB is deployed, and the MDB container invokes an application server-specific API that looks up the resource adapter by its JNDI name and invokes the specification-mandated `endpointActivation(MessageEndpointFactory, ActivationSpec)` method on the resource adapter.

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

8.3.2 Dispatching a Message

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

1. A message arrives from the EIS to the resource adapter.
2. The resource adapter examines the message and determines its type by looking it up in an internal table. The resource adapter determines the message type corresponds to a particular pair (MessageEndpointFactory, ActivationSpec).
3. The resource adapter determines the message should be dispatched to an MDB.
4. Using the MessageEndpointFactory for that type of message endpoint (one to be dispatched to an MDB), the resource adapter creates an MDB instance by invoking createEndpoint() on the factory.
5. The resource adapter then invokes the message listener method on the MDB instance, passing any required information (such as the body of the incoming message) to the MDB.
6. If the message listener does not return a value, the message dispatching process is complete.
7. If the message listener returns a value, the resource adapter determines how to handle that value. This may or may not result in further communication with the EIS, depending upon the contract with the EIS.

8.3.3 Activation Specifications

A resource adapter is configured with a mapping of message types and activation specifications. The activation specification is a JavaBean that implements javax.resource.spi.ActivationSpec. The resource adapter has an ActivationSpec class for each supported message type. The mapping of message types and activation specifications is configured in the ra.xml deployment descriptor, as described in Section 6.3, "Configuring Inbound Connections." For more information about ActivationSpecs, see Chapter 13, "Message Inflow," in JSR 322: Java EE Connector Architecture 1.6.

8.3.4 Administered Objects

As described in section 13.4.2.3 of JSR 322: Java EE Connector Architecture 1.6, a resource adapter may provide the Java class name and the interface type of an optional set of JavaBean classes representing administered objects that are specific to a messaging style or message provider. You configure administered objects in the admin-objects elements of the ra.xml and weblogic-ra.xml deployment descriptor files. As with outbound connections and other WebLogic resource adapter configuration elements, you can define administered objects at three configuration scope levels:

- Global - Specify parameters that apply to all administered objects in the resource adapter using the default-properties element. See default-properties in Table A-15, "admin-object-group"
Group - Specify parameters that apply to all administered objects belonging to a particular administered object group specified in the ra.xml deployment descriptor using the admin-object-group element. The properties specified in a group override any parameters specified at the global level. See Section A.6.1, "admin-object-group."

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

Instance - Under each admin object group, you can specify administered object instances using the admin-object-instance element of the weblogic-ra.xml deployment descriptor. These correspond to the individual administered objects for the resource adapter. You can use the admin-object-properties subelement to specify properties at the instance level too; properties specified at the instance level override those provided at the group and global levels. See Section A.6.1.1, "admin-object-instance."

8.4 Transactional Inflow

This section discusses how transactions flow into WebLogic Server from an EIS and a resource adapter. A transaction inflow contract allows the resource adapter to handle transaction completion and crash recovery calls initiated by an EIS. It also ensures that ACID properties of the imported transaction are preserved. For more information on transaction inflow, see Chapter 15, "Transaction Inflow," in JSR 322: Java EE Connector Architecture 1.6.

When an EIS passes a message through a resource adapter to the application server, it may pass a transactional context under which messages are delivered or work is performed. The inbound transaction will be controlled by a transaction manager external to the resource adapter and application server. See Section 8.3, "Message Inflow to Message Endpoints (Message-driven Beans)."

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

The WebLogic Server Transaction Manager is subordinate to the external Transaction Manager, which means that the external Transaction Manager ultimately determines whether the transaction succeeds or is rolled back. See "Participating in Transactions Managed by a Third-Party Transaction Manager" in Oracle Fusion Middleware Programming JTA for Oracle WebLogic Server. As part of the Java EE Connector Architecture 1.6, the ability for a resource adapter to participate in such a transaction is now exposed through a Java EE standard API.

The following process explains how a resource adapter would participate in a external transaction. For more information, see section 15.4, "Transaction Inflow Model," in JSR 322: Java EE Connector Architecture 1.6.

1. The resource adapter receives an inbound message with the transaction context that arrived along with the incoming message.
2. The resource adapter represents the transaction context using the `javax.transaction.xa.Xid` instance.

3. The resource adapter creates a new `Work` instance to process the incoming message and deliver it to a message endpoint, and also creates an instance of an `ExecutionContext (javax.resource.spi.work.ExecutionContext)`, setting the Xid it created and also setting a transaction timeout value. Version 1.6 of the Connector Architecture defines a standard class, `TransactionContext`, which resource adapters may use instead of the `ExecutionContext` for propagating the transaction context from the EIS to the application server.

4. The resource adapter submits the `Work` object and the `TransactionContext` (or `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.

   To use a `TransactionContext`, the `Work` class must:
   
   a. Implement the `javax.resource.spi.work.WorkContextProvider` interface.
   
   b. Create and return a `TransactionContext` instance in the `getWorkContexts()` method.

   **Note:** If the resource adapter uses a `TransactionContext`, the adapter must not use an `ExecutionContext`.

5. Subsequent XA calls from the external Transaction Manager are sent through the resource adapter and communicated to the WebLogic Server Transaction Manager. In this way, the resource adapter acts as a bridge for the XA calls between the external Transaction Manager and the WebLogic Server Transaction Manager, which is acting as a resource manager.

### 8.4.1 Using the Transactional Inflow Model for Locally Managed Transactions

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

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

### 8.5 Configuring and Managing Long-Running Work

As mentioned in Section 3.6, "Long-Running Work Support," WebLogic Server supports the use of `HintsContext.LONGRUNNING_HINT`. If set to `true` in a resource adapter `Work` class, `LONGRUNNING_HINT` causes a `Work` instance to be established as a long-running work item that WebLogic Server schedules in a separate daemon thread, not in a Work Thread. `LONGRUNNING_HINT` performs the same function as the WebLogic Server extension annotation `@LongRunning`.

WebLogic Server extends Connector Architecture 1.6 by providing the `ConnectorWorkManagerRuntimeMBean`, which contains attributes for configuring
and monitoring long-running Work instances. These attributes, described in the following sections, are also exposed in the WebLogic Server Administration Console.

- **Setting the Maximum Number of Concurrent Long-Running Work Instances**
- **Monitoring Long-Running Work**

For more information about the @LongRunning extension annotation, see LongRunning in Oracle Fusion Middleware Oracle WebLogic Server 10.3.1 API Reference.

### 8.5.1 Setting the Maximum Number of Concurrent Long-Running Work Instances

Oracle recommends that you minimize the number of long-running Work instances executing concurrently because each long running work runs in its own daemon thread. Having too many concurrent long-running Work instances can exhaust the thread resources in WebLogic Server and cause a negative impact on server performance and stability. WebLogic Server may introduce restrictions on maximum concurrent long running works allowed in a future release.

You can use the WebLogic Server Administration Console to set the maximum allowed number of concurrent Work instance requests as follows:

1. Select the resource adapter in the **Summary of Deployments > Control** page.
2. Select **Configuration > Workload**.
3. Enter a new value in **Maximum Number of Concurrent Long Running Requests**, if desired, and click **Save**.

   If you save a new value, the **Save Deployment Plan Assistant** is displayed, which prompts you to select or enter the path of a deployment plan file. For more information about working with deployment plans, see "Understanding WebLogic Server Deployment" in Deploying Applications to Oracle WebLogic Server.

Note the following:

- You can also view the maximum number of concurrent Work instance requests allowed from the Resource Adapter: Monitoring: Workload page in the Administration Console, as described in Section 8.5.2, "Monitoring Long-Running Work."

- As an alternative to using the Administration Console, you can use the max-concurrent-long-running-requests element in the weblogic-ra.xml file to set the maximum allowed number of concurrent Work instance requests. For information, see Section A.3, "connector-work-manager."

### 8.5.2 Monitoring Long-Running Work

The ConnectorWorkManagerRuntimeMBean exposes long-running run-time information about the resource adapter’s specific Work Manager in the following MBean attributes:

- **ConnectorWorkManagerRuntimeMBean.ActiveLongRunningRequests** — Returns the number of current active long-running Work instance requests.
- **ConnectorWorkManagerRuntimeMBean.CompletedLongRunningRequests** — Returns the number of completed long-running Work instance requests.
- **ConnectorWorkManagerRuntimeMBean.MaxConcurrentLongRunningRequests** — Returns the maximum number of concurrent Work instance requests allowed.
To view information about the currently active or completed long-running Work instance requests using the WebLogic Server Administration Console:

1. Select the resource adapter in the Summary of Deployments > Control page.

2. Select Monitoring > Workload.

The following information about long-running Work instance requests is available from the Long Running Work Managers table:

<table>
<thead>
<tr>
<th>The column labeled . . .</th>
<th>. . . identifies the following</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Requests</td>
<td>The number of currently active long-running Work instance requests.</td>
</tr>
<tr>
<td>Completed Requests</td>
<td>The number of completed long-running Work instance requests.</td>
</tr>
<tr>
<td>Max Concurrent Requests Allowed</td>
<td>The maximum number of concurrent Work instance requests allowed.</td>
</tr>
</tbody>
</table>
This chapter describes WebLogic Server resource adapter security for inbound and outbound communication. Resource adapters must be configured with authentication and other necessary security information to be able to establish connections with external systems.

This chapter includes the following sections:

- Container-Managed and Application-Managed Sign-on
- Credential Mapping for Making Outbound Connections
- Security Inflow
- Security Policy Processing
- Configuring Security Identities for Resource Adapters
- Configuring Connection Factory-Specific Authentication and Re-authentication Mechanisms

For more information about WebLogic security, see Understanding Security for Oracle WebLogic Server and Securing Resources Using Roles and Policies for Oracle WebLogic Server.

### 9.1 Container-Managed and Application-Managed Sign-on

When a resource adapter makes an outbound connection to an Enterprise Information System (EIS), it needs to sign on with valid security credentials. In accordance with JSR 322: Java EE Connector Architecture 1.6, WebLogic Server supports both container-managed and application-managed sign-on for outbound connections. At run time, WebLogic Server determines the chosen sign-on mechanism, based on the information specified in either the invoking client component’s deployment descriptor or the res-auth element of the resource adapter deployment descriptor. A sign-on mechanism specified in a resource adapter’s deployment descriptor takes precedence over one specified in the calling component’s deployment descriptor. Even when using container-managed sign-on, any security information explicitly specified by the client component is presented on the call to obtain the connection.

If the WebLogic Server Java EE 1.6 Connector Architecture implementation cannot determine which sign-on mechanism is being requested by the client component, the connector container attempts container-managed sign-on.

### 9.1.1 Application-Managed Sign-on

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

9.1.2 Container-Managed Sign-on

WebLogic Server and an EIS each maintain independent security realms. A goal of container-managed sign-on is to permit a user to sign on to WebLogic Server and be able to use applications that access an EIS through a resource adapter without having to sign on separately to the EIS. Container-managed sign-on in WebLogic Server uses **outbound credential mappings**, which map credentials (either username/password pairs or security tokens) of WebLogic security principals (which may be either authenticated individual users or client applications) to the corresponding credentials required to access the EIS. For any deployed resource adapter, you can configure outbound credential mappings for applicable security principals. For related information, see Section 9.2.2, "Outbound Credential Mappings."

9.2 Credential Mapping for Making Outbound Connections

JSR 322: Java EE Connector Architecture 1.6 requires storage of credentials in a javax.security.auth.Subject. The credentials are passed to either the createManagedConnection() or the matchManagedConnection() methods of the ManagedConnectionFactory object. Outbound credential mapping information is stored in the WebLogic Server embedded LDAP server. Outbound credential mappings are specific to outbound resource adapters.

When creating outbound credential mappings of WebLogic Server users to usernames in an EIS to which you want to connect using a resource adapter, note the following:

- WebLogic Server supports creating outbound credential mappings for WebLogic Server users who are defined in the default security realm only. If you are using a security realm that you have customized, you need to define it as the default security realm before configuring outbound credential mappings for resource adapters. For more information, see "Customizing the Default Security Configuration" in Securing Oracle WebLogic Server and "Change the default security realm" in Oracle WebLogic Server Administration Console Online Help.

- You must define the authentication-mechanism element for the connection pool in either of the following deployment descriptor files:
  - ra.xml, which works for all connection pools of the resource adapter
  - weblogic-ra.xml for each individual connection pool

If there is no valid authentication-mechanism element defined, the outbound credential mapping will not take effect, as explained in Section 9.2.1, "Authentication Mechanisms." The following is a sample ra.xml file snippet:

```xml
<authentication-mechanism>
  <authentication-mechanism-type>BasicPassword</authentication-mechanism-type>
  <credential-interface>javax.resource.spi.security.PasswordCredential</credential-interface>
</authentication-mechanism>
```

9.2.1 Authentication Mechanisms

WebLogic Server users must be authenticated whenever they request access to a protected WebLogic Server resource. For this reason, each user is required to provide a credential (a username/password pair or a digital certificate) to WebLogic Server.
Password authentication is the only authentication mechanism supported by WebLogic Server out of the box. Password authentication consists of a user ID and password. Based on the configured mappings, when a user requests connection to a resource adapter, the appropriate credentials for that user are supplied to the resource adapter.

The SSL (or HTTPS) protocol can be used to provide an additional level of security to password authentication. Because the SSL protocol encrypts the data transferred between the client and WebLogic Server, the user ID and password of the user do not flow in clear text. Using SSL, WebLogic Server can authenticate the user without compromising the confidentiality of the user’s ID and password. For more information, see “Configuring SSL” in Securing Oracle WebLogic Server.

### 9.2.2 Outbound Credential Mappings

Outbound credential mappings are specific to outbound resource adapters. You configure outbound credential mappings using the WebLogic Server Administration Console. Before you can configure credential mappings, you must successfully deploy the resource adapter.

**Note:** The first time you deploy a resource adapter, it has no configured outbound credential mappings and the initial connections will fail until they are configured.

If the resource adapter requires credentials and is configured to create connections at deployment time (meaning the initial-capacity element in the weblogic-ra.xml is set to greater than 0), the initial connection may fail. To prevent initial connection failure, Oracle recommends you set the initial-capacity element the connection pool to 0 for its connection pool for the initial installation and deployment of a resource adapter. Once the resource adapter is deployed for the first time, you can change the value of the initial-capacity element. For more information, see Section 6.4.1, “Initial-capacity: Setting the Initial Number of ManagedConnections.”

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

Review the following situations:

- **Non-initial Connection: Requires ManagedConnection from Adapter Upon Application’s Request**
- **Initial Connection: Requires a ManagedConnection from Adapter Without Application’s Request**
- **Special Users**

#### 9.2.2.1 Non-initial Connection: Requires ManagedConnection from Adapter Upon Application’s Request

WebLogic Server requires a ManagedConnection from the adapter upon an application’s request. For example, an application wants to get a connection from a pool but there is no available ManagedConnection in the pool so WebLogic Server needs to make a request to the adapter to create a new ManagedConnection.
The server searches for outbound mappings in the following order:

1. Specific mappings (or anonymous mapping if unauthenticated) at the connection factory level.
2. Specific mappings (or anonymous mapping if unauthenticated) at the global level.
3. Default mappings at the connection factory level.
4. Default mappings at the global level.

For example, in Example 9–1, consider two connection pools with the following outbound credential mappings:

**Example 9–1  Outbound Credential Mapping Examples**

**poolA**
- system user name: admin
- system password: adminpw
- default user name: guest1
- default password: guest1pw

**poolB**
- wlsjoe user name: harry
- wlsjoe password: harrypw

**global**
- system user name: sysman
- system password: sysmanpw
- wlsjoe user name: scott
- wlsjoe password: tiger
- default user name: viewer
- default password: viewerpw
- anonymous user name: foo
- anonymous password: bar

Referring to the example provided in Example 9–1, consider an application authenticated as system that makes a connection request against poolA. Because a specific outbound credential mapping is defined for system for poolA, the resource adapter uses this mapping (admin/adminpw).

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

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

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

1. The application searches at the global level, which also has no mapping for user1.
2. The application searches the poolA outbound mappings for a default mapping and finds a default mapping.
If an application does not authenticate to WebLogic Server and makes a request against poolA, it finds no outbound mapping for anonymous user for poolA. It then searches at the global level and finds a mapping for the anonymous user (foo/bar).

### 9.2.2.2 Initial Connection: Requires a ManagedConnection from Adapter Without Application’s Request

WebLogic Server requires a ManagedConnection from an adapter without the application’s request. This can either be when WebLogic Server creates initial connections at deployment time (meaning the *initial-capacity* element in `weblogic-ra.xml` is set to greater than 0), or when WebLogic Server needs to get a ManagedConnection specifically for XA recovery.

---

**Note:** Applies to both Container-Managed sign-on and Application-Managed sign-on.

---

WebLogic Server searches for outbound mappings in the following order:

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

If neither an initial nor default mapping is defined, WebLogic Server uses null as the Subject when making calls to the adapter to create a ManagedConnection.

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

**Example 9–2 Credential Mapping Examples**

```plaintext
poolA
  initial user name: admin
  initial password: adminpw

poolB
  default user name: harry
  default password: harrypw

global
  initial user name: sysman
  initial password: sysmanpw
```

Referring to **Example 9–2**, WebLogic Server needs to perform XA Recovery for poolA and so makes a connection request against poolA. Because the initial outbound credential mapping is defined for system for poolA, the resource adapter uses this mapping (admin/adminpw).

If WebLogic Server makes the same request against poolB, there is no corresponding initial outbound credential mapping for poolB. WebLogic Server then searches for the initial credential mapping at the global level where it finds a mapping (sysman/sysmanpw).

### 9.2.2.3 Special Users

Three special users are provided for use by resource adapters:
Initial User (user for creating initial connections) — If you define an outbound credential mapping for this user, the specified credentials are used for the initial connections created when:
- Starting the connection pool for this resource adapter
- Doing XA transaction recovery for the connection pool

The InitialCapacity parameter on the pool specifies the number of initial connections. If you do not define a mapping for this user, the default mapping (if provided) is used. Otherwise, no credentials are provided for the initial connections.

Anonymous User (unauthenticated WebLogic Server user) — If you define a mapping for this user, the specified credentials are used when no user is authenticated for the connection request on the resource adapter.

Default User — If you define a mapping for this user, the specified credentials are used when:
- No other mapping applies for the current user.
- No anonymous mapping is provided in the case where there is no authenticated user.

9.2.3 Creating Outbound Credential Mappings Using the Console

You can create outbound credential maps with the WebLogic Server Administration Console. If you are using the WebLogic Credential Mapping provider, the outbound credential maps are stored in the embedded LDAP server. For information about how to create an outbound credential map, see “Create outbound credential mappings” in the Oracle WebLogic Server Administration Console Online Help.

9.3 Security Inflow

JSR 322: Java EE Connector Architecture 1.6 defines a standard, generic security context between the EIS/resource adapter and the application server that leverages the work done in JSR 196: Java Authentication Service Provider Interface for Containers. The security context enables a resource adapter to establish security information that is used when submitting a Work instance for execution and delivering messages to message endpoints that are hosted in WebLogic Server.

JSR 322: Java EE Connector Architecture 1.6:
- Defines an abstract class SecurityContext as the contract between the resource adapter and the application server
- Defines two scenarios on how to handle flown-in identities based on whether or not they belong to the application server’s security domain:
  - Case 1 (see Section 16.4.3, "Case 1: Identity in the Container Security Domain," in JSR 322: Java EE Connector Architecture 1.6.)
  - Case 2 (Section 16.4.4, "Case 2: Identity Translated Between Security Domains.")
- Uses the CallbackHandler defined in the JSR 196: Java Authentication Service Provider Interface for Containers.
- Uses three callbacks from JSR 196: CallerPrincipalCallback, GroupPrincipalCallback, and PasswordValidationCallback.
9.3.1 Inbound Principal Mappings

A resource adapter deployed in the WebLogic Connector container can flow in an identity (that is, a caller principal, a group principal, or both) into a container, and the identity may be defined in either the WebLogic domain (as in the Case 1 scenario) or in the EIS security domain (as in the Case 2 scenario).

If the identity is defined in the EIS security domain, the WebLogic Connector container must be able to map the flown-in principal to a principal defined in the WebLogic domain. To support this scenario, WebLogic Server provides the ability to create an inbound principal mapping between the EIS principal and the corresponding WebLogic domain.

The following mappings can be created:

- Default mapping of EIS user names to either a specific WebLogic user, or the WebLogic user `anonymous`
- A specific EIS user name to either a specific WebLogic user, or the WebLogic user `anonymous`
- Default mapping of EIS group names to a WebLogic group name
- A specific EIS group name to a WebLogic group name

A principal name defined in an inbound principal mapping configuration must contain at least one non-space character. A string that contains only space characters is not a valid principal name (and is not accepted by the WebLogic Server Administration Console).

Note the following behavior regarding inbound principal mapping:

- Although JSR 322: Java EE Connector Architecture 1.6 allows a resource adapter to pass any user and group combination to the container, Connector Architecture 1.6 also allows the container to apply security restrictions. In the case of WebLogic Server, not all user and group combinations are valid: the WebLogic principals specified in the mapping must currently be defined in the WebLogic security realm, and the user must be defined in the WebLogic security realm as being a valid member of the group specified in the mapping. This is a requirement imposed by WebLogic Server.

For example, if a mapping specifies a particular user and group combination, and either the user or the group is not defined in the WebLogic Server security realm, the mapping is not valid. Additionally, if both the user and group are defined in the security realm, but the user is not a member of the particular group specified in the mapping, the mapping is not valid. When WebLogic Server determines that a mapping is not valid, it throws an exception.

Note also that WebLogic Server does not validate users or groups at the time an inbound principal mapping is configured. Because a security realm can be modified after the resource adapter has been deployed, WebLogic principals specified in an inbound principal mapping are validated only at run time.
A flown-in identity cannot run as a principal (that is, a user or group) that is mapped to an administrator role, such as Admin, AdminChannelUser, Deployer, Operator, or Monitor.

If no default inbound mapping is configured for an EIS user principal, and no mapping specific to the EIS user is configured, the user is mapped to an unauthenticated user.

If no default inbound mapping is configured for a EIS group principal, and no mapping specific to the EIS group is configured, the group principal is ignored.

Inbound principal mappings can be configured after the resource adapter has been deployed.

For information about how to create an inbound principal mapping using the WebLogic Server Administration Console, see “Create inbound principal mappings” in Oracle WebLogic Server Administration Console Online Help.

9.3.2 Security Inflow Callback Requirements

When a resource adapter flows in a identity that is used by the application server through handling CallerPrincipalCallback, GroupPrincipalCallback, and PasswordValidationCallback, JSR 322: Java EE Connector Architecture 1.6 does not specify any restrictions how those callbacks may be combined. However, not all combinations are valid in WebLogic Server Connector Architecture 1.6. The WebLogic Connector container validates these callbacks according to the requirements described in this section. You must design resource adapters so that they meet these requirements when they pass callbacks to the WebLogic Connector container. Otherwise, those callbacks are rejected.

WebLogic Server imposes the following requirements on callbacks passed to the Connector container:

- If a resource adapter handles a PasswordValidationCallback, the adapter must also handle a CallerPrincipalCallback. The WebLogic Security Service requires that a caller principal that is established by means of a CallerPrincipalCallback must match the user name that is authenticated by means of the PasswordValidationCallback.

- If a resource adapter handles a GroupPrincipalCallback, the adapter must also handle a CallerPrincipalCallback.

- A resource adapter must not handle a PasswordValidationCallback in Case 2 (see Section 16.4.4, "Case 2: Identity Translated Between Security Domains," in JSR 322: Java EE Connector Architecture 1.6). Because the username and password in the PasswordValidationCallback belong to the EIS security domain, the application server (that is, WebLogic Server) cannot authenticate them.

9.3.3 Backward Compatibility with Connector Architecture 1.5 and 1.0

WebLogic Server allows a resource adapter to use a configured principal to execute the Work.run() method. This principal can be configured in the WebLogic Server Administration Console, as described in “Configure security principals” in Oracle WebLogic Server Administration Console Online Help, or in the weblogic-ra.xml file using the run-work-as-principal-name and default-principal-name.

The Work.run() method then executes using the principal, if configured, or anonymous, by default, if this principal is not configured.
Configuring Security Identities for Resource Adapters

This mechanism provides a basic security configuration at the adapter level that applies to all Work instances submitted by the adapter. However, other security principals cannot be used selectively for different Work instances.

The security context feature in Connector Architecture 1.6 provides more flexibility by allowing each Work instance to provide its own SecurityContext, allowing each Work instance to be run under a different security principal.

Because the WebLogic Server Connector container is backward compatible with 1.0 and 1.5 adapters, note the following behavior when a resource adapter submits a Work instance:

- If the Work instance is submitted without a SecurityContext, the Work instance uses the principal defined in the run-work-as-principal-name and default-principal-name elements in the weblogic-ra.xml file.

- If the Work instance is submitted with a SecurityContext, the Work instance runs under the security principals that are defined according to the description of the SecurityContext class in JSR 322: Java EE Connector Architecture 1.6. The principals defined in the run-work-as-principal-name and default-principal-name elements, if present, are ignored.

9.4 Security Policy Processing

A security policy is an association between a WebLogic resource and one or more users, groups, or security roles and is designed to protect the WebLogic resource against unauthorized access. JSR 322: Java EE Connector Architecture 1.6 defines default security policies for resource adapters running in an application server. It also defines how resource adapters can provide their own specific security policies overriding the default. The weblogic.policy file that ships with WebLogic Server establishes the default security policies as specified in the Java EE Connector Architecture Specification.

If the resource adapter does not have a specific security policy defined, WebLogic Server establishes the runtime environment for the resource adapter with the default security policies specified in the weblogic.policy file, which conforms to the defaults specified by the Java EE Connector Architecture 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 21, "Runtime Environment," in JSR 322: Java EE Connector Architecture 1.6. For more information about security policies and the WebLogic security framework, see "Security Policies" in Securing WebLogic Resources Using Roles and Policies.

9.5 Configuring Security Identities for Resource Adapters

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

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

Example 9–3 is an excerpt from a `weblogic-ra.xml` deployment descriptor that illustrates how you would configure all four of these available security identities for performing different resource adapter tasks.

**Example 9–3 Configuring All Security Identities for Resource Adapters**

```xml
<weblogic-connector xmlns="http://xmlns.oracle.com/weblogic/weblogic-connector">
  <jndi-name>900blackbox-notx</jndi-name>
  <security>
    <default-principal-name>
      <principal-name>system</principal-name>
    </default-principal-name>
    <run-as-principal-name>
      <principal-name>raruser</principal-name>
    </run-as-principal-name>
    <run-work-as-principal-name>
      <principal-name>workuser</principal-name>
    </run-work-as-principal-name>
    <manage-as-principal-name>
      <principal-name>raruser</principal-name>
    </manage-as-principal-name>
  </security>
</weblogic-connector>
```

Example 9–4 illustrates how you could use the `default-principal-name` element to configure a single default principal security identity for performing all resource adapter tasks.

**Example 9–4 Configuring a Single Default Principal Identity for a Resource Adapter**

```xml
<weblogic-connector xmlns="http://xmlns.oracle.com/weblogic/weblogic-connector">
  <jndi-name>900blackbox-notx</jndi-name>
  <security>
    <default-principal-name>
      <principal-name>system</principal-name>
    </default-principal-name>
  </security>
</weblogic-connector>
```

For more information on setting security identity properties, see Section A.4, "security."

### 9.5.1 default-principal-name: Default Identity

You can define a single security identity that can be used for all resource adapter purposes using the `default-principal-name` element. If values are not specified for `run-as-principal-name`, `manage-as-principal-name`, and `run-work-as-principal-name`, they default to the value set for `default-principal-name`. 
The value of `default-principal-name` can be set to a defined WebLogic Server user name such as `system` or to use an anonymous identity (which is the equivalent of having no security identity).

For example, you can create a single security identity that makes all calls from WebLogic Server into the resource adapter and manages all resource adapter management tasks with a default `system` identity as shown in Example 9–6:

**Example 9–5  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:

**Example 9–6  Setting Up an Anonymous Identity**

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

### 9.5.2 `manage-as-principal-name`: Identity for Running Management Tasks

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

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

**Example 9–7** illustrates how you can configure a resource adapter to run management calls using the WebLogic Server-defined user name `system`.

**Example 9–7  Using a Defined WebLogic Server Name**

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

**Example 9–8** illustrates how you can configure a resource adapter to run management calls using an anonymous identity.

**Example 9–8  Setting Up an Anonymous Identity**

```
<security>
  <manage-as-principal-name>
    <use-anonymous-identity>true</use-anonymous-identity>
  </manage-as-principal-name>
</security>
```
9.5.3 run-as-principal-name: Identity Used for Connection Calls from the Connector Container into the Resource Adapter

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

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

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

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

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

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

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

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

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

To run work using the requesting caller's identity, you specify the run-work-as-principal-name element as shown in Example 9–9:

**Example 9–9 Using the Requesting Caller's Identity**

```xml
<security>
  <run-work-as-principal-name>
    <use-caller-identity>true</use-caller-identity>
  </run-work-as-principal-name>
</security>
```
9.6 Configuring Connection Factory-Specific Authentication and Re-authentication Mechanisms

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

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

You can override the `authentication-mechanism` and `reauthentication-support` values in the `ra.xml` deployment descriptor by specifying them in the `weblogic-ra.xml` deployment descriptor. Doing so allows you to apply these settings to a specific connection factory rather than to all connection factories. See `authentication-mechanism` and `reauthentication-support` in Table A–18, “default-connection-properties subelements”.
This chapter describes how to package WebLogic resource adapters (RARs) and deploy them in an exploded directory format or as an archive file.

This chapter includes the following sections:

- Section 10.1, "Packaging Resource Adapters,"
- Section 10.2, "Deploying Resource Adapters,"

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

### 10.1 Packaging Resource Adapters

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

#### 10.1.1 Packaging Directory Structure

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

Resource adapters use the same directory format, whether the resource adapter is packaged in an exploded directory format or as a RAR. A typical directory structure of a resource adapter is shown in Example 10–1:

**Example 10–1 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
10.1.2 Packaging Considerations

The following are packaging requirements for resource adapters:

- Deployment descriptors (ra.xml and weblogic-ra.xml) must be in a directory called META-INF.
- An optional MANIFEST.MF also resides in META-INF. A manifest file is automatically generated by the JAR tool and is always the first entry in the JAR file. By default, it is named META-INF/MANIFEST.MF. The manifest file is the place where any meta-information about the archive is stored.
- A resource adapter deployed in WebLogic Server supports the class-path entry in MANIFEST.MF to reference a class or resource such as a property.
- The resource adapter can contain multiple JARs that contain the Java classes and interfaces used by the resource adapter. (For example, eis.jar and utilities.jar.) Ensure that any dependencies of a resource adapter on platform-specific native libraries are resolved.
- The resource adapter can contain native libraries required by the resource adapter for interacting with the EIS. (For example, windows.dll and unix.so.)
- The resource adapter can include documentation and related files not directly used by the resource adapter. (For example, readme.html and /images/ra.jpg.)
- When a standalone resource adapter RAR is deployed, the resource adapter must be made available to all Java EE applications in the application server.
- When a resource adapter RAR packaged within a Java EE application EAR is deployed, the resource adapter must be made available only to the Java EE application with which it is packaged. This specification-compliant behavior may be overridden if required.

10.1.3 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 JSR 322: Java EE Connector Architecture 1.6 limitation of not providing a remotable interface.

10.1.4 Packaging Resource Adapter Archives (RARs)

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

---

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

---

To stage and package a resource adapter:

1. Create a temporary staging directory anywhere on your hard drive.
2. Compile or copy the resource adapter Java classes into the staging directory.
3. Create a JAR to store the resource adapter Java classes. Add this JAR to the top level of the staging directory.

4. Create a META-INF subdirectory in the staging directory.

5. Create an ra.xml deployment descriptor in the META-INF subdirectory and add entries for the resource adapter.

---

**Note:** Refer to the following document for information about the ra.xml document type definition:

---

6. Create a weblogic-ra.xml deployment descriptor in the META-INF subdirectory and add entries for the resource adapter.

---

**Note:** Refer to Appendix A, "weblogic-ra.xml Schema" for information on the contents of the weblogic-ra.xml file.

---

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

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

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

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

   For more information on this topic, see Section 3.1, "Creating and Configuring Resource Adapters: Main Steps."

10.2 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.

10.2.1 Deployment Options

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

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

For information about these application deployment techniques, see "Deploying Applications and Modules with weblogic.deployer" in Deploying Applications to Oracle WebLogic Server.

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

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

10.2.2 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.

10.2.3 Production Redeployment

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

10.2.3.1 Suspendable Interface and Production Redeployment

Typically, a resource adapter bean implements the
javax.resource.spi.ResourceAdapter interface. This interface defines
start() and stop() methods. This type of resource adapter is not eligible for
production redeployment. Resource adapters connect to one or more EISes for
incoming/outgoing communication. All communication is performed in a resource
adapter-proprietary way with no knowledge of the application server. If on-the-fly
production redeployment is attempted, the application server can only provide
notifications to the resource adapter to manage the migration of connections from the
existing resource adapter to a new instance. However, the resource adapter can
implement the Suspendable interface, which provides the capability to allow
resource adapters to participate in production redeployment. For information about
implementing the Suspendable interface, see Section 4.4, "Suspending and
Resuming Resource Adapter Activity."
10.2.3.2 Production Redeployment Requirements

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

- The resource adapter must be based on Connector Architecture 1.6. (Support for production redeployment of 1.0 resource adapters is not available.)
- The resource adapter must implement the `Suspendable` interface (see Example 4–3).
- The resource adapter must be packaged inside an enterprise application (EAR file). Production redeployment of standalone resource adapters is not supported.
- The `Suspendable.supportsVersioning()` method must return `true` when invoked by WebLogic Server.
- The `enable-access-outside-app` element in the `weblogic-ra.xml` descriptor must be set to `false`.

10.2.3.3 Production Redeployment Process

The following process assumes the older version of the resource adapter is deployed and running. It also assumes that the older version (named `old`) as well as the newer version (named `new`) of the resource adapter meet all of the requirements mentioned in Section 10.2.3.2, "Production Redeployment Requirements," as well as the application requirements described in "Redeploying Applications in a Production Environment" in Deploying Applications to Oracle WebLogic Server.

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

1. WebLogic Server calls `new.init(old, null)` to inform the new resource adapter that it is replacing the old resource adapter.
2. WebLogic Server calls `old.startVersioning(new, null)` to inform the old resource adapter to start its production redeployment operation with the new resource adapter.
3. WebLogic Server calls `new.start(extendedBootstrapContext)`. See Section 4.5, "Extended BootstrapContext".
4. When the old resource adapter is "finished" (meaning it has succeeded in migrating all clients and inbound connections to the new resource adapter), it calls `(ExtendedBootstrapContext)bsCtx.complete()`. This informs WebLogic Server that it is safe to undeploy the old resource adapter.
5. When undeployment occurs, WebLogic Server calls `old.stop()` and production redeployment is complete.

The calls to `new.init()` and `old.startVersioning()` give the old and new resource adapters an opportunity to migrate inbound or outbound communications from the old to the new resource adapter. How this is done is up to the individual resource adapter developer.
The following sections in this appendix describe the deployment descriptor elements that can be defined in the WebLogic Server-specific deployment descriptor weblogic-ra.xml. The schema for weblogic-ra.xml is http://xmlns.oracle.com/weblogic/weblogic-connector/1.3/weblogic-connector.xsd. If your resource adapter archive (RAR) does not contain a weblogic-ra.xml deployment descriptor, WebLogic Server automatically selects the default values of the deployment descriptor elements.

- weblogic-connector
- work-manager
- connector-work-manager
- security
- properties
- admin-objects
- outbound-resource-adapter

### A.1 weblogic-connector

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>native-libdir</td>
<td>Required if native libraries are present.</td>
<td>Specifies the directory where all the native libraries exist that are required by the resource adapter.</td>
</tr>
<tr>
<td>jndi-name</td>
<td>Required only if a resource adapter bean is specified.</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table A–1  (Cont.) weblogic-connector subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
</table>
| enable-access-outside-app      | Optional          | As stated by JSR 322: Java EE Connector Architecture 1.6, 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  
Note: When set to false, the resource adapter can only be accessed by clients that reside within the same application in which the resource adapter resides.  
For version 1.0 resource adapters (supported in this release), the default value for this element is set to true. |
| enable-global-access-to-classes| Optional          | When set to true, the resource adapter allows global access to its classes, and the adapter's classes are loaded by the WebLogic Server system classpath classloader directly so that these classes can be accessed by all applications.  
The default value is false, in which case the adapter's classes are loaded by a classloader that is a child of the EAR's application classloader.  
This value normally should be set to true for standalone adapters.  
When set to true, you must to restart WebLogic Server if you change the adapter's classes and want to redeploy the adapter. |
| work-manager                   | Optional          | This complex element is used to specify all the configurable elements for creating the Work Manager that will be used by the resource adapter bean. The work-manager element is imported from the weblogic-javaee.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 Section A.2, "work-manager." |
| connector-work-manager         | Optional          | This complex element is used to specify all the configurable elements for the Connector Work Manager for this adapter module itself.  
This element provides configurations that are not supported by the standard Work Manager.  
For subelements of connector-work-manager, see Section A.3, "connector-work-manager." |
| security                       | Optional          | This complex element is used to specify all the security parameters for the operation of the resource adapter.  
See Section A.4, "security," for information on the security defaults that will be taken by the connector container. |
properties

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 Section A.5, "properties."

admin-objects

This complex element defines all of the admin objects in a resource adapter. As with the outbound-resource-adapter complex element (see Section A.7, "outbound-resource-adapter"), the admin-objects complex element has four hierarchical property levels that specify the configuration scope:

1. Global level — at this level, you specify parameters that apply to all admin objects in the resource adapter; you do so using the default-properties element. See default-properties in Table A–14, "admin-objects subelements".

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 Section A.6.1, "admin-object-group."

3. Instance level — under each admin object group, you can use the admin-object-instance element to specify admin object instances. These correspond to the admin object instances for the resource adapter. You can specify properties at the instance level and override those properties provided in the group and global levels. See Section A.6.1.1, "admin-object-instance."

For admin-objects subelements, see Section A.6, "admin-objects."

Table A–1 (Cont.) weblogic-connector subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>Optional</td>
<td>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 Section A.5, &quot;properties.&quot;</td>
</tr>
<tr>
<td>admin-objects</td>
<td>Optional</td>
<td>This complex element defines all of the admin objects in a resource adapter. As with the outbound-resource-adapter complex element (see Section A.7, &quot;outbound-resource-adapter&quot;), the admin-objects complex element has four hierarchical property levels that specify the configuration scope:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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 in Table A–14, &quot;admin-objects subelements&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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 Section A.6.1, &quot;admin-object-group.&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Instance level — under each admin object group, you can use the admin-object-instance element to specify admin object instances. These correspond to the admin object instances for the resource adapter. You can specify properties at the instance level and override those properties provided in the group and global levels. See Section A.6.1.1, &quot;admin-object-instance.&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For admin-objects subelements, see Section A.6, &quot;admin-objects.&quot;</td>
</tr>
</tbody>
</table>
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-javaee.xsd` schema. The following subelements can be configured in the `work-manager` element.

### Table A–2的工作管理器子元素

<table>
<thead>
<tr>
<th>元素</th>
<th>必选/可选</th>
<th>描述</th>
</tr>
</thead>
</table>
| name                      | 必选     | 指定工作管理器的名称。JSR 322: Java EE 连接器架构 1.6 描述如何使用资源适配器将工作线程提交给应用服务器。这些工作线程由 WebLogic Server 工作管理器管理。工作管理器动态调整工作线程数量以避免死锁并实现最佳吞吐量，同时满足并发约束。它还满足响应时间目标、份额和优先级。

#### A.2 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-javaee.xsd` schema. The following subelements can be configured in the `work-manager` element.

### Table A–2 work-manager subelements

<table>
<thead>
<tr>
<th>元素</th>
<th>必选/可选</th>
<th>描述</th>
</tr>
</thead>
</table>
| outbound-resource-adapter| 可选     | 这个复杂元素用于描述资源适配器的 outbound 组件。与 `admin-objects` 复杂元素不同，此元素具有三层分层属性，用于定义 outbound 连接池：

1. **全局层次** — 在此级别，您可以指定适用于资源适配器中所有 outbound 连接池的参数。使用 `default-connection-properties` 元素。参见“A.7.1, "default-connection-properties."”

2. **群组层次** — 在此级别，您可指定适用于特定连接工厂在 `ra.xml` 部署描述符中指定的 outbound 连接的参数。使用 `connection-definition-group` 元素。`connection-definition-group` 元素与 `weblogic-ra.xml` 中的连接定义组具有一对一的对应关系。群组中指定的属性将覆盖全局层次指定的任何参数。参见“A.7.2, "connection-definition-group."”

3. **实例层次** — 在每个连接定义组下，您可以指定连接实例。这些对应于资源适配器的个别连接池。可以在实例层次指定参数，这些参数将覆盖群组和全局层次指定的参数。参见“A.7.2.1, "connection-instance."”

对于 `outbound-resource-adapter` 子元素，参见“Section A.7, "outbound-resource-adapter."”
The connector-work-manager element is a complex element that is used to specify all the configurable elements for the Connector Work Manager for the resource adapter module. This element provides configurations that are not supported by the standard WebLogic Work Manager. The following subelement can be configured in the connector-work-manager element.

### Table A–2 (Cont.) work-manager subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>response-time-request-class</td>
<td>Optional</td>
<td>A work-manager element can include one and only one of the following four elements: response-time-request-class - Defines the response time request class for the application. Response time is defined with attribute goal-ms in milliseconds. The increment is ((goal - T) Cr)/R, where T is the average thread use time, R the arrival rate, and Cr a coefficient to prioritize response time goals over fair shares. fair-share-request-class - Defines the fair share request class. Fair share is defined with attribute percentage of default share. Therefore, the default is 100. The increment is Cf/(P R T), where P is the percentage, R the arrival rate, T the average thread use time, and Cf a coefficient for fair shares to prioritize them lower than response time goals. context-request-class - Defines the context class. Context is defined with multiple cases mapping contextual information, like current user or its role, cookie, or work area fields to named service classes. request-class-name - Defines the request class name.</td>
</tr>
<tr>
<td>fair-share-request-class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context-request-class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>request-class-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>min-threads-constraint</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>min-threads-constraint-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max-threads-constraint</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>max-threads-constraint-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>capacity</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>capacity-name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The security complex element contains default security information that can be configured for the connector container. For more information, see Section 9.5, "Configuring Security Identities for Resource Adapters."

### Table A–3 connector-work-manager subelement

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-concurrent-long-running-requests</td>
<td>Optional</td>
<td>Specifies the maximum number of concurrent long-running Work instance requests allowed for a resource adapter instance. Because each long-running Work instance request executes in its own thread, an excessive number of long-running Work requests can have a negative affect on server performance and stability. A resource adapter typically needs only a few long-running Work requests, such as periodically listening to a socket or scheduling other Work instances. New long-running Work request submissions are rejected if the number of currently active long-running Work requests exceeds the specified limit. Default value: 10</td>
</tr>
</tbody>
</table>

### Table A–4 security subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-principal-name</td>
<td>Optional</td>
<td>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 Section A.4.1, &quot;default-principal-name&quot; for subelements of this element.</td>
</tr>
<tr>
<td>manage-as-principal-name</td>
<td>Optional</td>
<td>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 Section A.4.2, &quot;manage-as-principal-name&quot; for subelements of this element.</td>
</tr>
<tr>
<td>run-as-principal-name</td>
<td>Optional</td>
<td>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 Section A.4.3, &quot;run-as-principal-name&quot; for subelements of this element.</td>
</tr>
</tbody>
</table>
A.4.1 default-principal-name

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-anonymous-identity</td>
<td>Required</td>
<td>Specifies that the anonymous identity should be used.</td>
</tr>
<tr>
<td>principal-name</td>
<td>Required</td>
<td>Specifies that the principal name should be used. This should match a defined WebLogic Server user name.</td>
</tr>
</tbody>
</table>

A.4.2 manage-as-principal-name

The `manage-as-principal-name` element contains the following subelements.
Table A–6  manage-as-principal-name subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-anonymous-identity</td>
<td>Required</td>
<td>Specifies that the anonymous identity should be used.</td>
</tr>
<tr>
<td>principal-name</td>
<td>Required</td>
<td>Specifies that the principal name should be used. This should match a defined WebLogic Server user name.</td>
</tr>
</tbody>
</table>

A.4.3 run-as-principal-name

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

Table A–7  run-as-principal-name subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-anonymous-identity</td>
<td>Required</td>
<td>Specifies that the anonymous identity should be used.</td>
</tr>
<tr>
<td>principal-name</td>
<td>Required</td>
<td>Specifies that the principal name should be used. This should match a defined WebLogic Server user name.</td>
</tr>
<tr>
<td>use-caller-identity</td>
<td>Required</td>
<td>Specifies that the caller’s identity should be used.</td>
</tr>
</tbody>
</table>

A.4.4 run-work-as-principal-name

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

Table A–8  run-work-as-principal-name subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-anonymous-identity</td>
<td>Required</td>
<td>Specifies that the anonymous identity should be used.</td>
</tr>
<tr>
<td>principal-name</td>
<td>Required</td>
<td>Specifies that the principal name should be used. This should match a defined WebLogic Server user name.</td>
</tr>
<tr>
<td>use-caller-identity</td>
<td>Required</td>
<td>Specifies that the caller’s identity should be used.</td>
</tr>
</tbody>
</table>

A.4.5 security-work-context

The security-work-context element contains the following subelements.
The caller-principal-default-mapped element contains the following subelements.

Table A–10 caller-principal-default-mapped subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-anonymous-identity</td>
<td>Required</td>
<td>Specifies that the WebLogic anonymous user identity should be used. Note that you can choose either use-anonymous-identity or principal-name, but not both.</td>
</tr>
<tr>
<td>principal-name</td>
<td>Required</td>
<td>Specifies that the principal name should be used. This should match a WebLogic user name defined in the WebLogic security realm.</td>
</tr>
</tbody>
</table>
A.4.5.2 caller-principal-mapping
The caller-principal-mapping complex element is used to specify a mapping from an EIS group name to WebLogic group name. It contains the following subelements.

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eis-caller-principal</td>
<td>Required</td>
<td>Specifies an EIS user principal name.</td>
</tr>
<tr>
<td>mapped-caller-principal</td>
<td>Required</td>
<td>Specifies either the mapped WebLogic user principal name or the anonymous user identity (but not both).</td>
</tr>
</tbody>
</table>

A.4.5.3 group-principal-mapping
The group-principal-mapping element contains the following subelements.

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eis-group-principal</td>
<td>Required</td>
<td>Specifies an EIS group principal name.</td>
</tr>
<tr>
<td>mapped-group-principal</td>
<td>Required</td>
<td>Specifies the mapped WebLogic group principal name.</td>
</tr>
</tbody>
</table>

A.5 properties
The properties element, a subelement of weblogic-connector, is a container for properties specified for the resource adapter bean in ra.xml. It holds one more or more property elements.

You define property elements within the properties element as follows.

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>Required</td>
<td>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.</td>
</tr>
</tbody>
</table>

A.6 admin-objects
The admin-objects complex element defines all of the admin objects in the resource adapter. As with the outbound-resource-adapter complex element, the
admin-objects complex element has three hierarchical property levels that you can specify.

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

### Table A–14 admin-objects subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-properties</td>
<td>Optional</td>
<td>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 Section A.5, &quot;properties.&quot;</td>
</tr>
<tr>
<td>admin-object-group</td>
<td>One or more</td>
<td>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 Section A.6.1, &quot;admin-object-group&quot;</td>
</tr>
</tbody>
</table>

#### A.6.1 admin-object-group

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

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

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

### Table A–15 admin-object-group

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin-object-interface</td>
<td>Required</td>
<td>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.</td>
</tr>
</tbody>
</table>
A.6.1.1 admin-object-instance

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

Table A–16 admin-object-instance subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jndi-name</td>
<td>Required</td>
<td>The JNDI name used to define the reference name for the connection instance. The connection pool is bound into a JNDI that clients outside the application can see. <strong>Note:</strong> The enable-access-outside-app element must be set to true.</td>
</tr>
<tr>
<td>admin-object-properties</td>
<td>Optional</td>
<td>Defines all the properties that apply to the admin object instance. The admin-object-properties element can contain one or more property elements, each holding a name and value pair. See Section A.5, &quot;properties.&quot;</td>
</tr>
</tbody>
</table>

A.7 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.
A.7.1 default-connection-properties

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

Table A–17  outbound-resource-adapter subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default-connection-properties</td>
<td>Optional</td>
<td>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 Section A.7.1, &quot;default-connection-properties.&quot;</td>
</tr>
</tbody>
</table>

Table A–18  default-connection-properties subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool-params</td>
<td>Optional</td>
<td>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 Section A.7.1.1, &quot;pool-params.&quot;</td>
</tr>
<tr>
<td>logging</td>
<td>Optional</td>
<td>Contains parameters for configuring logging of the ManagedConnectionFactory and ManagedConnection objects of the resource adapter. For subelements, see Section A.7.1.2, &quot;logging.&quot;</td>
</tr>
</tbody>
</table>
### A.7.1.1 pool-params

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

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction-support</td>
<td>Optional</td>
<td>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 6, “Connection Management.”</td>
</tr>
<tr>
<td>authentication-mechanism</td>
<td>Optional</td>
<td>The <code>authentication-mechanism</code> 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.</td>
</tr>
<tr>
<td>reauthentication-support</td>
<td>Optional</td>
<td>A Boolean that specifies whether a particular connection factory supports re-authentication of an existing ManagedConnection instance. It provides the ability to override the reauthentication-support value specified in the ra.xml deployment descriptor that is intended to be the default value for all Connection Factories of the resource adapter.</td>
</tr>
<tr>
<td>properties</td>
<td>Optional</td>
<td>The <code>properties</code> element includes one or more property elements, which define name and value subelements that apply to the default connections.</td>
</tr>
<tr>
<td>res-auth</td>
<td>Optional</td>
<td>Specifies whether to use container- or application-managed security. The values for this element can be one of Application or Container. The default value is Container.</td>
</tr>
</tbody>
</table>
### Table A–19  pool-params subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial-capacity</td>
<td>Optional</td>
<td>Specifies the initial number of ManagedConnections, which WebLogic Server attempts to create during deployment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 1</td>
</tr>
<tr>
<td>max-capacity</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 10</td>
</tr>
<tr>
<td>capacity-increment</td>
<td>Optional</td>
<td>Specifies the maximum number of additional ManagedConnections that WebLogic Server attempts to create during resizing of the maintained connection pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 1</td>
</tr>
<tr>
<td>shrinking-enabled</td>
<td>Optional</td>
<td>Specifies whether unused ManagedConnections will be destroyed and removed from the connection pool as a means to control system resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: true</td>
</tr>
<tr>
<td>shrink-frequency-seconds</td>
<td>Optional</td>
<td>Specifies the amount of time (in seconds) the Connection Pool Management waits between attempts to destroy unused ManagedConnections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 900 seconds</td>
</tr>
<tr>
<td>highest-num-waiters</td>
<td>Optional</td>
<td>Specifies the maximum number of threads that can concurrently block waiting to reserve a connection from the pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 0</td>
</tr>
<tr>
<td>highest-num-unavailable</td>
<td>Optional</td>
<td>Specifies the maximum number of ManagedConnections in the pool that can be made unavailable to the application for purposes such as refreshing the connection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note that in cases like the backend system being unavailable, this specified value could be exceeded due to factors outside the pool’s control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 0</td>
</tr>
<tr>
<td>connection-creation-retry-frequency-seconds</td>
<td>Optional</td>
<td>The periodicity of retry attempts by the pool to create connections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 0</td>
</tr>
<tr>
<td>connection-reserve-timeout-seconds</td>
<td>Optional</td>
<td>Sets the number of seconds after which the call to reserve a connection from the pool will timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: -1 (do not block when reserving resources)</td>
</tr>
<tr>
<td>test-frequency-seconds</td>
<td>Optional</td>
<td>The frequency with which connections in the pool are tested.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 0</td>
</tr>
</tbody>
</table>
A.7.1.2 logging

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

Table A–20 logging subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-filename</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>logging-enabled</td>
<td>Optional</td>
<td>Indicates whether or not the log writer is set for either the ManagedConnectionFactory or ManagedConnection. If this element is set to true, output generated from either the ManagedConnectionFactory or ManagedConnection will be sent to the file specified by the log-filename element. Default Value: false</td>
</tr>
</tbody>
</table>
### Table A–20 (Cont.) logging subelements

<table>
<thead>
<tr>
<th>Element</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotation-type</td>
<td>Optional</td>
<td>Sets the file rotation type. Possible values are bySize, byTime, none.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bySize - When the log file reaches the size that you specify in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>file-size-limit, the server renames the file as FileName.n.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>byTime - At each time interval that you specify in file-time-span,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the server renames the current log file. After the server renames a file,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subsequent messages accumulate in a new file with the name that you</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specified in log-filename.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>none - Messages accumulate in a single file. You must erase the contents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of the file if the log size becomes unwieldy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: bySize</td>
</tr>
<tr>
<td>number-of-files-limited</td>
<td>Optional</td>
<td>Specifies whether to limit the number of files that this server instance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>creates to store old log messages. (Requires that you specify a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rotation-type of bySize or byTime). After the server reaches this limit,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>it overwrites the oldest file. If you do not enable this option, the server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>creates new files indefinitely and you must clean up these files as you</td>
</tr>
<tr>
<td></td>
<td></td>
<td>require.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you enable number-of-files-limited by setting it to true, the server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>refers to your rotationType variable to determine how to rotate the log file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rotate means that you override your existing file instead of creating a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>new file. If you specify false for number-of-files-limited, the server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>creates numerous log files rather than overriding the same one.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: false</td>
</tr>
<tr>
<td>file-count</td>
<td>Optional</td>
<td>The maximum number of log files that the server creates when it</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rotates the log. This number does not include the file that the server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses to store current messages. (Requires that you enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>number-of-files-limited.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 7</td>
</tr>
<tr>
<td>file-size-limit</td>
<td>Optional</td>
<td>The size that triggers the server to move log messages to a separate file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Requires that you specify a rotation-type of bySize.) After the log file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reaches the specified minimum size, the next time the server checks the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>file size, it will rename the current log file as FileName.n and create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a new one to store subsequent messages.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 500</td>
</tr>
<tr>
<td>rotate-log-on-startup</td>
<td>Optional</td>
<td>Specifies whether a server rotates its log file during its startup cycle.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: true</td>
</tr>
<tr>
<td>log-file-rotation-dir</td>
<td>Optional</td>
<td>Specifies the directory path where the rotated log files will be stored.</td>
</tr>
<tr>
<td>rotation-time</td>
<td>Optional</td>
<td>The start time for a time-based rotation sequence of the log file, in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format k:mm, where k is 1-24. (Requires that you specify a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rotation-type of byTime.) At the specified time, the server renames the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>current log file. Thereafter, the server renames the log file at an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interval that you specify in file-time-span.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the specified time has already past, then the server starts its file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rotation immediately.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By default, the rotation cycle begins immediately.</td>
</tr>
<tr>
<td>file-time-span</td>
<td>Optional</td>
<td>The interval (in hours) at which the server saves old log messages to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>another file. (Requires that you specify a rotation-type of byTime.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default Value: 24</td>
</tr>
</tbody>
</table>
A.7.2 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 connection-definition-interface element.

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

<table>
<thead>
<tr>
<th>Table A–21 connection-definition-group subelements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>connection-factory-interface</td>
</tr>
<tr>
<td>default-connection-properties</td>
</tr>
<tr>
<td>connection-instance</td>
</tr>
</tbody>
</table>

A.7.2.1 connection-instance

You can define the following subelements under connection-instance.

<table>
<thead>
<tr>
<th>Table A–22 connection-instance subelements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>description</td>
</tr>
<tr>
<td>jndi-name</td>
</tr>
<tr>
<td>Element</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>connection-properties</td>
</tr>
</tbody>
</table>
This appendix describes some best practices for resource adapter developers.

- Classloading Optimizations for Resource Adapters
- Connection Optimizations
- Thread Management
- InteractionSpec Interface
- Using javax.jms.ConnectionFactory

### B.1 Classloading Optimizations for Resource Adapters

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

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

To avoid such performance issues, you can do either of the following:

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

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

### B.2 Connection Optimizations

Oracle recommends that resource adapters implement the optional enhancements described in sections 7.16.1 and 7.16.2 of JSR 322: Java EE Connector Architecture 1.6. Implementing these interfaces allows WebLogic Server to provide several features that will not be available without them.

Lazy Connection Association Optimization, as described in section 7.16.1, allows the server to automatically clean up unused connections and prevent applications from hogging resources. Lazy Transaction Enlistment Optimization, as described in 7.16.2, allows applications to start a transaction after a connection is already opened.
B.3 Thread Management

Resource adapter implementations should use the WorkManager (as described in Chapter 10, "Work Management," in JSR 322: Java EE Connector Architecture 1.6) 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.

B.4 InteractionSpec Interface

WebLogic Server supports the Common Client Interface (CCI) for EIS access, as defined in Chapter 17, "Common Client Interface," in JSR 322: Java EE Connector Architecture 1.6. The CCI defines a standard client API for application components that enables application components and EAI frameworks to drive interactions across heterogeneous EISes.

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

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

This is not the case for standalone resource adapters (packaged as RAR files). If the interface is serializable (as is the case with InteractionSpec), then both the client and the resource adapter need access to the InteractionSpec interface as well as the implementation classes. However, if the interface extends java.io.Remote, then the client only needs access to the interface class.

B.5 Using javax.jms.ConnectionFactory

When using an EJB or servlet to send messages using a JCA adapter backing a JMS provider using XA transactions, the resource-ref needs to be java.lang.object.

In a WebLogic Server environment, specifying javax.jms.ConnectionFactory implements WebLogic JMS Wrappers which are not compatible with this JCA adapter configuration. See "Enhanced Support for Using WebLogic JMS with EJBs and Servlets" in Oracle Fusion Middleware Programming JMS for Oracle WebLogic Server.