

Jakarta Connectors Guide

JEUS 9.1

TMAXSOFT

Copyright

Copyright 2025. TmaxSoft Co., Ltd. All Rights Reserved.

Company Information

TmaxSoft Co., Ltd.

TmaxSoft Tower, 45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, South Korea

Website: <https://www.tmaxsoft.com/en/>

Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd. Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features.

This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

Java, Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Microsoft, Windows, Windows NT are registered trademarks or trademarks of Microsoft Corporation.

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

AIX is a registered trademark of International Business Machines Corporation.

UNIX is a registered trademark of X/Open Company, Ltd.

Linux is a registered trademark of Linus Torvalds.

Noto is a trademark of Google Inc. Noto fonts are open source. All Noto fonts are published under the SIL Open Font License, Version 1.1. (<https://www.google.com/get/noto/>)

Other products and company names are trademarks or registered trademarks of their respective

owners.

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (™, ®).

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory:
\${INSTALL_PATH}/license/oss_licenses

Document History

Product Version	Guide Version	Date	Remarks
JEUS 9.1	3.3.1	2025-12-10	-
JEUS 9	3.1.2	2025-03-24	-
JEUS 9	3.1.1	2024-12-24	-

Table of Contents

1. JCA Standards	1
1.1. Overview	1
1.2. Connector Architecture	1
1.2.1. Outbound	2
1.2.2. Inbound	3
1.3. Resource Adapters and JDBC Driver	4
1.4. CCI(Common Client Interface)	4
2. Outbound Management	6
2.1. Managing the Connection Pool and Connections	6
2.1.1. Connection Request Pattern of an Application	6
2.1.2. Relationship between Resource Adapter and Connection Pool	7
2.1.3. Strength of Connection Pool	8
2.1.4. Connection Pool Configuration	8
2.1.5. Connection Pool Features	12
2.2. Managing Transactions	14
2.2.1. Local Transaction Resource Participating in Global Transaction (XA)	14
2.2.2. Global Transaction (XA) and Connection Sharing	14
2.3. Example of Connection Pool Configuration	15
2.3.1. Configuring a Connection Pool with One Connection Factory	15
2.3.2. Configuring a Connection Pool with Two Connection Factories	16
2.4. Monitoring and Controlling Connection Pools	18
2.4.1. Controlling Connection Pools	19
2.4.2. Monitoring Connection Pools	20
3. Inbound Management	23
3.1. Managing the Work Manager	23
3.1.1. Basic Concepts	23
3.1.2. Work Manager Configuration	23
3.2. Message Inflow	24
4. Resource Adapter	27
4.1. Security Management	27
4.1.1. Connection Authentication	27
4.2. Packaging	28
4.3. Deploy	28
4.3.1. Class Loading in SHARED Mode	29
4.3.2. Redeploy	29
4.4. Registering Resource Adapters as Resources	29
Appendix A: Notes for jeus-connector-dd.xml Configuration	31

1. JCA Standards

This chapter describes JCA standards and resource adapters.

1.1. Overview

Jakarta™ Connectors (JCA) provide a standardized architecture for connecting the Jakarta EE platform with Enterprise Information Systems (EIS).

Originally introduced in J2EE 1.3, JCA was designed to offer a scalable and secure method for integrating Web Application Server (WAS) and Jakarta EE applications running on WAS with EIS. Since its introduction, JCA has been continuously enhanced with new features.

1.2. Connector Architecture

The Connector architecture is the standard defined for interoperating with the Enterprise Information System (EIS) including but not limited to the mainframe, Enterprise Resource Planning (ERP), TP Monitor, and the legacy database systems. Without the standard, a separate custom driver needs to be implemented between each EIS and Web Application Server (WAS) vendors, resulting in a problem with N by M complexity. This will significantly impair the portability and scalability of the Jakarta EE environment.

To tackle such problem, the JCA standard introduces a concept of connector architecture and defines Resource Adapter needed for interoperation. If EIS vendors provide resource adapters, using the connector architecture, they can run on multiple WASs without code modification. For more information about resource adapters, refer to [Resource Adapters and JDBC Driver](#).



This guide does not cover information about the connector architecture. Refer to related documents or JCA 2.0 standards for more information.

There are two kinds of connector architectures; outbound and inbound. 'Out' and 'in' refer to communication direction between WAS and EIS.

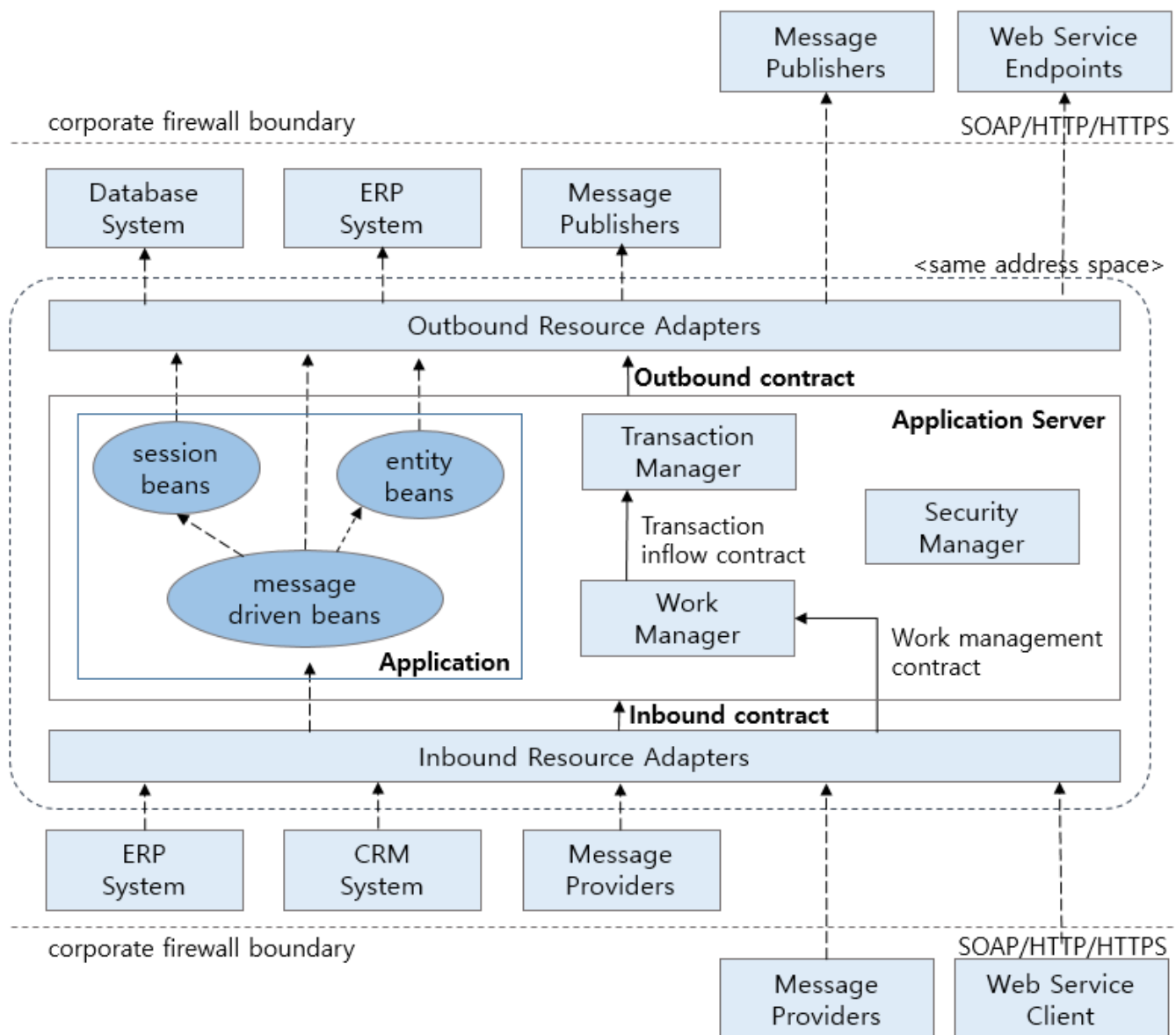
- Outbound

From applications that were deployed on WAS to EIS.

- Inbound

From EIS to applications that were deployed on WAS.

In the connector architecture, the roles of WAS and resource adapter for outbound and inbound communications is defined.



Overview of the Connector Architecture

1.2.1. Outbound

A term that represents the outbound architecture is 'connection'. Applications running on WAS create connections to send requests and receive results.

A typical example is sending SQL from servlet or EJB component to the database. In the figure [Overview of the Connector Architecture](#), outbound corresponds to the flow of a request from Session Beans or Entity Beans to the EIS through the resource adapter.

The following is provided for efficient management of connections.

- Connection Pool
 - Enhances the performance efficiency by reusing connections without creating a new connection each time.
 - Keeps the number of requests from WAS to resources from increasing infinitely when the load is heavy.

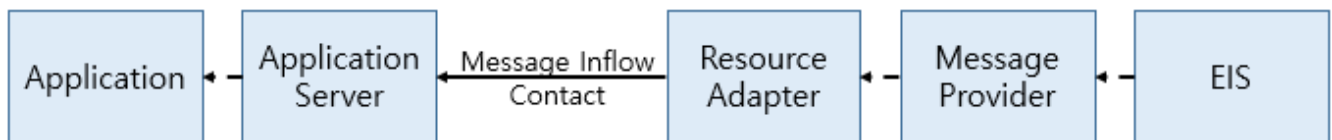
- Transaction Management
 - Automatically connects with the transaction manager if a global transaction (XA) is needed.
 - Supports connection sharing for global transactions (XA).

Besides what is mentioned, JEUS also provides additional functionalities. For other functions of outbound, refer to [Outbound Management](#).

1.2.2. Inbound

The JCA standard defines a mechanism for interoperating with various kinds of message-oriented services including Jakarta Message Service (JMS). In the mechanism, a resource adapter plays the role of the message provider, and an end-point application that of the message consumer. The type of a transmitted message has no relationship with WAS, and the resource adapter can arbitrarily decide on a message interface that defines a method for the actual transmission of the messages.

The following figure shows the message inflow.



Inbound Message Inflow

A message provider sends a message only by a direct request from a message consumer. No context is transmitted and shared between them. In order to send information related to a transaction that occurred in EIS or authentication to WAS, it is necessary to transmit the context through the Work Manager.

For details about inbound communication management of JEUS, refer to [Inbound Management](#) or [Message Inflow](#).



For details about context transmission, refer to the JCA standards.

Work Manager

A resource adapter processing outbound messages does not require a thread, but the one processing inbound messages may require a thread. The JCA standard recommends using a WAS-provided Work Manager instead of creating a Java thread. In particular, the Work Manager must be used to interoperate with a transaction that started in EIS or with authentication related information.

The implementation entity of JEUS Work Manager is based on the thread pool. If a resource adapter sends a Work Instance, the thread pool obtains a worker thread to process the instance. The resource adapter can send an event listener to the worker thread to receive the start and end process events and exception occurrences. It can also send the required transaction and authentication information as a context.

For more information about the implementation entity of JEUS Work Manager, refer to [Managing the Work Manager](#). For basic concept about Work Manager, see the JCA standards.

1.3. Resource Adapters and JDBC Driver

While JDBC defines a standard for the interoperation with relational database (RDB), resource adapter defines standard for multiple information systems including RDB. The resource adapter defines WAS-interoperable external resources more generally and widely than the JDBC driver.

The big difference between a resource adapter and a JDBC driver is that the resource adapter is defined in the Jakarta EE standard, while the JDBC driver is built based on the Java SE standard. Therefore, the JDBC driver can be used by a stand-alone Java application.

A JDBC driver uses the system classpath of a jar file, and to update the driver currently running JVM needs to be shut down. The resource adapter, on the other hand, is an application defined by Jakarta EE, it can be packaged into an RAR file and deployed to WAS.

To update the version of a resource adapter, redeploy it without terminating WAS. It is also possible, like the JDBC driver, to convert the adapter into a jar file and configure it using the system classpath, but in general, a resource adapter is a Jakarta EE application that can be redeployed.

The following table compares resource adapters to JDBC drivers.

	Resource Adapter	JDBC Driver
Concept	Defined in the Jakarta EE standard.	Java SE concept. (JDBC driver can be used directly in a stand-alone Java application.)
Interoperation	Defines a standard for interoperation with several information systems including relational database (RDB).	Defines a standard for communication with relational database (RDB).
Deployment	Can be packaged into a rar file, and deployed to WAS.	Uses system classpath of a jar file.
Update	Redeployment without terminating WAS.	For an update, shut down the running JVM.

Compared to the JDBC driver, the resource adapter, defined more generally and widely, can be used more easily.

1.4. CCI(Common Client Interface)

Common Client Interface (CCI) provides outbound resource adapter as an integrated API. The JCA standard recommends using CCI when developing a resource adapter.

CCI conceptualizes a service call as an interaction, and abstracts the service method as an instance dependent on an EIS called interaction Spec. Arguments and returned values of the call is provided in

the form of a record.

In general, it is recommended to include the `FunctionName` and `InteractionVerb` in the interaction specification. CCI supports three types of services -synchronized transmission, synchronized reception, and synchronized transmission and reception, but does not support asynchronous services.



CCI is derived from an intent to define a standard interface between an application and a resource adapter. In general, WAS is not involved with the workings of the interface between an application and resource adapter, but considers it as a `java.lang.Object` type.

The following is the preparation steps for calling an EIS service using CCI.

```
// get Connection to EIS by lookup ConnectionFacotry in JNDI
javax.naming.Context nc = new InitialContext();
jakarta.resource.cci.ConnectionFactory cf =
(ConnectionFactory)nc.lookup("java:/comp/env/eis/ConnectionFactory");
jakarta.resource.cci.Connection cx = cf.getConnection();

// create Interaction
jakarta.resource.cci.Interaction ix = cx.createInteraction();

// create Interaction Spec
com.wombat.cci.InteractionSpecImpl spec = .....
spec.setFunctionName("<EIS_FUNCTION_NAME>");
spec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE)"
. . .
```

The data that the EIS service needs is saved as a record, and the service is called as follows:

```
// call EIS service
boolean ret = ix.execute(spec, input, output);
```



For details about CCI, refer to the JCA standard or CCI related documents.

2. Outbound Management

This chapter describes roles and functions of JEUS in an outbound communication from an application to EIS, based on interoperation with a connection pool and transaction.

2.1. Managing the Connection Pool and Connections

A connection is required for an application, in an outbound communication, to send a request to EIS and receive a response. JEUS provides the connection pool for an application to efficiently use the connection.

2.1.1. Connection Request Pattern of an Application

An application can request a connection Factory, and then obtain a connection from the factory. This is the same as the method for looking up a data source to obtain a JDBC connection, and then obtaining a connection from the data source.

The following is an example of connection request pattern of an application.

```
// obtain the initial JNDI Naming context
Context initctx = new InitialContext();
// perform JNDI lookup to obtain the connection factory
jakarta.resource.cci.ConnectionFactory connectionFactory =
    (jakarta.resource.cci.ConnectionFactory)initctx.lookup("java:comp/env/eis/sampleEIS");
jakarta.resource.cci.Connection conn = connectionFactory.getConnection();
try {
    // do some works
} finally {
    conn.close();
}
```



In the example, CCI is used, but a connection factory does not actually have a specified interface. It is defined by a resource adapter and used by an application.

Assuming that the application above is an EJB, the annotation or **<resource-ref>** in the ejb-jar.xml file should be configured as follows:

Connection request: <ejb-jar.xml>

```
<ejb-jar>
. . .
<enterprise-beans>
. . .
  <session>
    . . .
    <resource-ref>
```

```

    <res-ref-name>eis/sampleEIS</res-ref-name>
    <res-type>
        jakarta.resource.cci.ConnectionFactory
    </res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
. . .

```

In jeus-ejb-dd.xml, a JNDI name to which 'eis/sampleEIS' is to be mapped should be specified.

In the following example, 'sampleConnectionPool', is the JNDI name of the connection pool that is specified in jeus-connector-dd.xml when deploying a resource adapter.

Connection request: <jeus-ejb-dd.xml>

```

<jeus-ejb-dd>
. . .
  <beanlist>
    <jeus-bean>
      . . .
      <res-ref>
        <jndi-info>
          <ref-name>eis/sampleEIS</ref-name>
          <export-name>sampleConnectionPool</export-name>
        </jndi-info>
      </res-ref>
    . . .
  </beanlist>
. . .

```



For more information about jeus-connector-dd.xml, refer to [Resource Adapter](#).

2.1.2. Relationship between Resource Adapter and Connection Pool

If an application requests a connection factory through JNDI, JEUS requests a resource adapter to create the connection factory.

To do so, JEUS passes a connection manager to the resource adapter. The implementation entity of the connection manager is the **JEUS connection pool**.

The resource adapter will pass the created connection factory to JEUS, and JEUS will pass it to the application. If the application requests the connection factory to create a Connection, the resource adapter requests the connection manager for a connection. The JEUS connection manager then takes a physical connection (or ManagedConnection) from the connection pool to create a connection handle, and returns it to the resource adapter. The resource adapter passes it to the application. A physical connection and connection handle are defined in the JCA standard.

If a connection is no longer needed, the application has to return it. This is called 'closing the connection.' If the application does not properly close a connection, there will be a connection leak.

It is the resource adapter's choice to use the connection manager passed from JEUS. If a resource adapter creates a connection internally without using a connection manager, JEUS is not involved in connection management and communication between an application and EIS.



1. For information about the connection manager, refer to the Javadoc and JCA standards for `jakarta.resource.spi.ConnectionManager` and `jakarta.resource.spi.ManagedConnectionFactory`.
2. For further information about physical connection and connection handle, refer to "Chapter 6. Connection Management" of the JCA standards.

2.1.3. Strength of Connection Pool

The functions of JEUS connection pool provided by JCA are similar to those of DB connection pool but have the following advantages.

- High performance

In general, a process for creating a connection to a database or external resource is slow. However, since a physical connection in the connection pool can be reused without having to recreate it each time, overhead for creating a connection can be reduced. The overhead for disconnecting can also be reduced, as a connection closed by an application is not actually disconnected, but returned to the pool.

- Simultaneous connection management

The number of clients sending requests to a resource can be controlled. This prevents too many requests from being sent to the resource and causing an outage.



For more information about DB Connection Pool, refer to "Database Connection Pool and JDBC" in *JEUS Server Guide*.

2.1.4. Connection Pool Configuration

JEUS connection pool can be configured in the `jeus-connector-dd.xml` file when deploying a resource adapter.

JCA connection pool is automatically created without being configured. In order for an application to use a connection pool, a JNDI name is required, which is automatically created according to the rule set by JEUS. In order for a user to specify a JNDI name, it should be set in `jeus-connector-dd.xml` file. A connection pool can be configured inside the `<jeus-connector-dd>` tag, which is inside the `<connection-pool>` tag, in `jeus-connector-dd.xml` file. For information about how to include the `jeus-connector-dd.xml` file in RAR, refer to [Resource Adapter](#).

By the JCA standard, multiple connection pools can be configured to one resource adapter. A

connection pool is based on <connection-definition> element in ra.xml file, and its sub-element, <connectionfactory-interface>. For example, if five separate <connection-definition> elements are specified, five connection pools are created in JEUS.



<connectionfactory-interface> defined in ra.xml file cannot be repeated. If the same value is repeated in ra.xml file, the validity check will fail, and deployment will also fail.

The following is an example of the jeus-connector-dd.xml file. For more information about each tag, refer to [Connection Pool Configuration](#).

- **<connectionfactory-interface>**

- If ra.xml file contains two or more <connection-definition> elements, <connectionfactory-interface> element should be specified. Otherwise, it is impossible to create a connection pool, and a deployment error will occur.
- Specify a value for the <connectionfactory-interface> element specified inside each <connection-definition> element in ra.xml file. For details about <connectionfactory-interface>, refer to [Resource Adapter](#).

- **<export-name>**

- JNDI name of a connection pool.
- The name should be unique within the server, and it is required.

- **<transaction-support>**

- Specify a transaction type supported by the connection pool. The specified value overrides the configuration in ra.xml file.
 - NoTransaction
 - LocalTransaction
 - XATransaction

- **<user>**

- User ID sent to the connection pool to create a Connection.

- **<password>**

- Password sent to the connection pool to create a Connection.
- To store an encrypted password, configure it as follows:

```
{algorithm}ciphertext
```

Example:

```
{DES}FQrLbQ/D801LDVS71L28rw==
```

- **<use-lazy-transaction-enlistment>**

- Determines whether to use the "Lazy Transaction Enlistment" option, which is one of the transaction optimization functions stated in the JCA standard. (Default value: false)

- **<pool-management>**

Main options of a connection pool exist inside the <pool-management> sub-element of the <connection-pool> element.

- **<min>**: Initial value for the number of connections (Default value: 2)
- **<max>**: The maximum number of connections allowed in a connection pool (Default value: 10)
- **<step>**: The number of connections added when it needs to be increased. (Default value: 1)
- **<period>**: Sets the size of a connection pool to the minimum value at specified intervals. If there is no idle connection in the pool, the size is not reduced to the minimum value. The unit is in milliseconds. This setting replaces the <pooled-timeout> value. (Default value: 10 minutes, Unit: ms)
- **<wait-connection>**: Determines how to process a connection request that is issued when there is no idle connection and the size of the connection pool reaches the maximum value.

Tag	Description
<wait-connection>	<ul style="list-style-type: none">◦ true: The system will wait to obtain a valid connection. If it fails to obtain a connection during the wait time, an exception occurs.◦ false: Provide the application with a newly created connection and when the application returns it, close the connection without saving it in the pool. This is called a 'disposable connection' in JEUS. (Default value)
<wait-timeout>	Only valid when <wait-connection> is set to true. Specifies how long a user should wait for a connection. If no connection is available for the user during the time period, an exception occurs. (Default value: 10 seconds, Unit: ms)

- **<use-match-connection>**: Determines whether to use Connection Match. (Default value: false)
- **<allow-disposable-connection-when-match-failed>**: Determines whether to use a disposable connection if Connection Match fails. If Connection Match is not set, this setting is not used. (Default value: false)
- **<connection-validation>**: Sets a validity check for a connection.

Tag	Description
<enabled>	Determines whether to use the connection validity check function. Even if this tag is set to true, the resource adapter must implement the jakarta.resource.spi.ValidatingManagedConnectionFactory interface to use this setting.

Tag	Description
<period>	Specifies an interval for the connection validity check. Checks the validity of idle connections at specified intervals. (Unit: ms)
<non-validation-interval>	If the gap between the last time the connection was used and the start time of the validity check is smaller than the specified value, validity check is not performed. This setting can reduce the overhead of the check. (Unit: ms)
<validation-retrial-count>	<p>When the destroy policy is set to FailedConnectionOnly, the validity check is performed only once. When the policy is set to AllConnections, the check is performed twice for each connection, and another one if the first one fails.</p> <p>If more validity checks are needed, the number can be increased using this setting.</p>
<destroy-policy-on-validation>	<p>Determines how to process connections in the connection pool when the connection validity check fails.</p> <ul style="list-style-type: none"> ◦ FailedConnectionOnly: Only closes the physical connection for which the validity check failed. (Default value) ◦ AllConnections: If the validity check fails, it will be performed for another connection in the pool. If the check fails again, all connections in the pool will be closed. This includes all connections that are being used by the application.

- **<action-on-connection-leak>**: Sets logging or return action for connections used by a component (Mainly stateless components- servlet/JSP, stateless session beans, and MDB). If not set, the default setting for Action On Resource Leak configured in the server will be performed. (Default value: Warning)
- **<connection-trace>**: Option to monitor connections. The default setting is that the stack trace during getConnection is displayed in order to show which applications are using the connections. Such information is also displayed when Action On Resource Leak configured in the server is processed.

Option	Description
enabled	Turn the connection trace function on/off.
get-connection-trace	Stores stack trace information when an application obtains a connection (calling getConnection). (Default value: true)
local-transaction-trace	<p>Determines whether to trace information about local transactions between an application and a resource adapter.</p> <p>When used with the <get-connection-trace> setting, it is helpful for tracing an application that did not properly commit or roll back a local transaction.</p>

- **<max-use-count>**: A connection can be used as many times as the specified value, and after that, it is closed and a new connection is created. (Default value: 0, meaning that the connections will continue to be used without being replaced).
- **<pool-destroy-timeout>**: Waiting time to destroy a connection pool. A connection pool is destroyed when undeploying the resource adapter. Doing network communication with the resource while closing connections may cause the system to hang. If destruction is not executed during the specified waiting time, undeploy will continue to be processed ignoring the setting. (Default value: 10 seconds)
- **<property>**
 - Adds a property to be applied to ManagedConnectionFactory. Used to replace or add a value specified in the ra.xml file.



For more information about Connection Match and Lazy Transaction Enlistment, refer to the JCA standard.

2.1.5. Connection Pool Features

A connection pool can examine the validity of connections, and handle connection leaks.

Connection Validity Check

When an application requests a connection, a connection pool can request a resource adapter to check validity of a connection. This is useful for checking disconnection caused by an internal error of the connection, or socket disconnection caused by the firewall. If validity check fails, a new physical connection is created and returned to the application.

If validity check is performed too often incurring overhead, configure the **<non-validation-interval>** setting. If the gap between the last time the connection was used and the start time of the validity check is less than the specified value, the validity check is not performed since the connection is considered valid in this case.

For example, if an interval is set to 5 seconds (5,000 ms), and less than five seconds have passed since the connection was last used, the connection is assumed to be valid.

The following is an example of the **<non-validation-interval>** element.

```
<non-validation-interval>5000</non-validation-interval>
```



To execute the validity check, the resource adapter has to first implement the `jakarta.resource.spi.ValidatingManagedConnectionFactory` interface. Sometimes, when validity check is requested to a resource adapter, there may be no response. Since WAS does not currently have a solution for this, a timeout option

should be provided by the resource adapters.

If validity check fails, the user can specify a destroy policy for the other connections in the connection pool. The policy is described in the following table.

Policy	Description
FailedConnectionOnly	Only discards a connection for which validity check failed. (Default value)
AllConnections	Discards all the other connections along with the failed connection. Before discarding all the connections after the validity check fails, another check is attempted. If it fails again, all connections in the pool are discarded.

The following is an example of setting the Destroy policy for connections in the pool when the validity check fails.

```
<destroy-policy-on-validation>AllConnections</destroy-policy-on-validation>
```



To increase the number of connection validity checks executed, use `<validation-retrial-count>`.

Handling Connection Leak

JEUS can specify an action to handle connection leaks in each connection pool or server.

For components such as servlet/JSP, stateless session beans, and message driven bean, whose start and end are clear, it is easy to check if a used connection was returned properly. If a connection has not been returned, it is possible to set an action to force the connection to be closed, or record logs through the Invocation Manager. However, a JCA connection throws an exception for the automatic return (AutoClose) action of the Invocation Manager.

Since from the JCA connection pool it is not possible to know which method closes a connection, the pool cannot directly close the connection. However, if the `<connection-interface>` element inside the `<connection-definition>` element in resource adapter's `ra.xml` file is configured as the following, the pool is able to close the connection. This is because, in this case, the pool already knows the structure of the 'close' method and the method has no parameters.

The following are the exception handling methods for the AutoClose action of JCA connection's Invocation Manager.

- `java.sql.Connection`
- `jakarta.resource.cci.Connection`

If the previous `<connection-interface>` element is not configured, it is difficult to directly close a connection. However, after calling the cleanup method defined in

jakarta.resource.spi.ManagedConnection, the connection is forced to be returned to the pool.

In the 'cleanup' method, the resource adapter's role is defined as follows:

The cleanup should invalidate all connection handles that had been created using this ManagedConnection instance.



For more information about Invocation Manager, refer to "Action On Resource Leak" in *JEUS Server Guide*.

2.2. Managing Transactions

This section describes how JEUS manages a connection pool whose transaction type is either local or global.

2.2.1. Local Transaction Resource Participating in Global Transaction (XA)

Since a local transaction is a transaction between an application and resource adapter, JEUS cannot request a resource adapter for a connection unless it's for a global transaction.

When using a local transaction type connection pool for a global transaction, jakarta.resource.spi.LocalTransaction instance obtained from the ManagedConnection instance of a resource adapter can be emulated as an XAResource object.

Even if a resource adapter does not support global transactions (XA), if it supports local transactions, at most one resource can participate in a global transaction (XA). This works like the connection pool data source that uses the XA emulation function.



For detailed information on data sources, refer to "Data Source Configuration" in *JEUS Server Guide*.

2.2.2. Global Transaction (XA) and Connection Sharing

Connection sharing means that the same connection is always used for a particular resource within the same global transaction (XA). JEUS supports connection sharing, and connections can be shared without any settings.

To disable the connection sharing function, for each application, configure the **<resource-ref>** element as Unsharable.

<resource-ref>

```
<res-ref-name>jca/pool</res-ref-name>
<res-type>jakarta.resource.cci.ConnectionFactory</res-type>
<res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```



1. For details about connection sharing, refer to the section "8.9 Connection Sharing" of the JCA standard 2.0.
2. DB Connection Pool also provides Connection Sharing. Refer to "Global Transaction (XA) and Connection Sharing" in *JEUS Server Guide*.

2.3. Example of Connection Pool Configuration

One outbound resource adapter can be used to configure one or more outbound connections.

2.3.1. Configuring a Connection Pool with One Connection Factory

The following is an example of the ra.xml file that contains jakarta.resource.cci.ConnectionFactory interface as the Connection Factory element. In most cases, there is only one Connection Factory specified as shown in the following.

Configuring a connection pool with one connection factory: <ra.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="https://jakarta.ee/xml/ns/jakartaee" version="2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
    https://jakarta.ee/xml/ns/jakartaee/connector_2_0.xsd">
  <display-name>ConnectionManagementSample1</display-name>
  <vendor-name>TmaxSoft</vendor-name>
  <eis-type>TestResource</eis-type>
  <resourceadapter-version>2.0</resourceadapter-version>
  <license>
    <license-required>false</license-required>
  </license>
  <resourceadapter>
    <outbound-resourceadapter>
      <connection-definition>
        <managedconnectionfactory-class>
          com.tmax.SampleManagedConnectionFactory
        </managedconnectionfactory-class>
        <connectionfactory-interface>
          jakarta.resource.cci.ConnectionFactory
        </connectionfactory-interface>
        <connectionfactory-impl-class>
          com.tmax.SampleConnectionFactory
        </connectionfactory-impl-class>
        <connection-interface>
          jakarta.resource.cci.Connection
        </connection-interface>
        <connection-impl-class>
```

```

        com.tmax.SampleConnection
    </connection-impl-class>
</connection-definition>
<transaction-support>
    LocalTransaction
</transaction-support>
<authentication-mechanism>
    <authentication-mechanism-type>
        BasicPassword
    </authentication-mechanism-type>
    <credential-interface>
        jakarta.resource.spi.security.PasswordCredential
    </credential-interface>
</authentication-mechanism>
<reauthentication-support>
    false
</reauthentication-support>
</outbound-resourceadapter>
</resourceadapter>
</connector>

```

If the jeus-connector-dd.xml file is not configured, you cannot create a connection pool corresponding to the <connection-definition> setting in the ra.xml file. Therefore, you must configure the **<export-name>** element in the jeus-connector-dd.xml file.

The following is an example of the jeus-connector-dd.xml file. For more information about each tag, refer to [Connection Pool Configuration](#).

Configuring a connection pool with one connection factory: <jeus-connector-dd.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <connection-pool>
        <export-name>jcapool</export-name>
        <pool-management>
            <min>10</min>
            <max>50</max>
            <period>600000</period> <!-- 10 minutes -->
            <wait-connection>
                <wait-connection>true</wait-connection>
                <wait-timeout>30000</wait-timeout>
            </wait-connection>
            <action-on-connection-leak>Warning</action-on-connection-leak>
        </pool-management>
    </connection-pool>
</jeus-connector-dd>

```

2.3.2. Configuring a Connection Pool with Two Connection Factories

The following is an example of the ra.xml file that configures javax.sql.DataSource and jakarta.resource.cci.ConnectionFactory interfaces as Connection Factory settings.

Configuring a connection pool with two connection factories: <ra.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="https://jakarta.ee/xml/ns/jakartaee" version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
        https://jakarta.ee/xml/ns/jakartaee/connector_2_0.xsd">
    <display-name>ConnectionManagementSample1</display-name>
    <vendor-name>TmaxSoft</vendor-name>
    <eis-type>TestResource</eis-type>
    <resourceadapter-version>2.0</resourceadapter-version>
    <license>
        <license-required>false</license-required>
    </license>
    <resourceadapter>
        <outbound-resourceadapter>
            <connection-definition>
                <managedconnectionfactory-class>
                    com.tmax.DataSourceManagedConnectionFactory
                </managedconnectionfactory-class>
                <connectionfactory-interface>
                    javax.sql.DataSource
                </connectionfactory-interface>
                <connectionfactory-impl-class>
                    com.tmax.JeusDataSource
                </connectionfactory-impl-class>
                <connection-interface>
                    java.sql.Connection
                </connection-interface>
                <connection-impl-class>
                    com.tmax.JeusConnection
                </connection-impl-class>
            </connection-definition>
            <connection-definition>
                <managedconnectionfactory-class>
                    com.tmax.SampleManagedConnectionFactory
                </managedconnectionfactory-class>
                <connectionfactory-interface>
                    jakarta.resource.cci.ConnectionFactory
                </connectionfactory-interface>
                <connectionfactory-impl-class>
                    com.tmax.SampleConnectionFactory
                </connectionfactory-impl-class>
                <connection-interface>
                    jakarta.resource.cci.Connection
                </connection-interface>
                <connection-impl-class>
                    com.tmax.SampleConnection
                </connection-impl-class>
            </connection-definition>
            <transaction-support>
                NoTransaction
            </transaction-support>
            <authentication-mechanism>
                <authentication-mechanism-type>
                    BasicPassword
                </authentication-mechanism-type>
                <credential-interface>
                    jakarta.resource.spi.security.PasswordCredential
                </credential-interface>
            </authentication-mechanism>
        </outbound-resourceadapter>
    </resourceadapter>
</connector>
```

```

        </credential-interface>
    </authentication-mechanism>
    <reauthentication-support>
        false
    </reauthentication-support>
</outbound-resourceadapter>
</resourceadapter>
</connector>

```

As shown in the following, you need to configure the **<connectionfactory-interface>** element to create a connection pool through the jeus-connector-dd.xml file according to the <connection-definition> setting of the ra.xml file. Otherwise, an error that a connection pool cannot be created occurs during deployment.

Configuring a connection pool with two connection factories: <jeus-connector-dd.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <connection-pool>
        <export-name>jdbcpool</export-name>
        <connectionfactory-interface>
            javax.sql.DataSource
        </connectionfactory-interface>
        <pool-management>
            <min>5</min>
            <max>20</max>
        </pool-management>
    </connection-pool>
    <connection-pool>
        <export-name>ccipool</export-name>
        <connectionfactory-interface>
            jakarta.resource.cci.ConnectionFactory
        </connectionfactory-interface>
        <pool-management>
            <min>1</min>
            <max>10</max>
        </pool-management>
    </connection-pool>
</jeus-connector-dd>

```

2.4. Monitoring and Controlling Connection Pools

Use the JEUS console tool (jeusadmin) to monitor and control the JCA Connection Pool. To do so, resource adapter modules must be deployed first. For more information about how to deploy resource adapter modules, refer to *JEUS Applications & Deployment Guide*.

The examples in this section assume that a resource adapter module, whose JNDI export name of JCA connection pool is 'my_rar_cp,' has already been deployed. For JNDI export name configuration of JCA connection pool, refer to [Connection Pool Configuration](#).



For more information about monitoring and control with console tools, refer to

2.4.1. Controlling Connection Pools

The following shows how to control connection pools using jeusadmin.

```
[MASTER]domain1.adminServer>cpinfo
The connection pool information on the server [adminServer].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID |     |     |        |     |         |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| mysql *| 20  | 20  | 0       | 0   | 0.0   | 0    | 0      | 0  | fal  | false|
|         |     |     |         |     |        |      |        |   | se   |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| oracle | 1   | 30  | 0       | 0   | 0.0   | 0    | 0      | 0  | true | false|
| *      |     |     |         |     |        |      |        |   |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
=====
[MASTER]domain1.adminServer>create-connection-pool -id mysql
Servers that successfully created a connection pool : adminServer
Servers that failed to create a connection pool : none.
[MASTER]domain1.adminServer>cpinfo
The connection pool information on the server [adminServer].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID |     |     |        |     |         |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| mysql  | 20  | 20  | 1       | 0   | 0.0   | 20   | 0      | 20 | fal  | true |
|         |     |     |         |     |        |      |        |   | se   |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| oracle | 1   | 30  | 0       | 0   | 0.0   | 0    | 0      | 0  | true | false|
| *      |     |     |         |     |        |      |        |   |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
=====
[MASTER]domain1.adminServer>control-connection-pool -id mysql -enable
Servers that successfully enabled a connection pool : adminServer
Servers that failed to enable a connection pool : none.
[MASTER]domain1.adminServer>
```

First, since the Connection Pool has not been created yet, create a Connection Pool with the create-connection-pool command. After that, you can control the Connection Pool with the control-connection-pool command.

The following describes the options for using control-connection-pool.

Button	Description
[Enable]	Activates the connection pool.
[Shrink]	Adjusts the number of connections in the connection pool to the minimum value.
[Disable]	Inactivates the connection pool.
[Refresh]	Replaces connections of the connection pool with new connections.

2.4.2. Monitoring Connection Pools

The following shows how to monitor connection pools using jeusadmin.

```
[MASTER]domain1.adminServer>cpinfo
The connection pool information on the server [adminServer].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID |     |     |        |     |        |      |        |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| mysql  | 20  | 20  | 1      | 0   | 0.0   | 20   | 0      | 20  | fal  | true |
|        |     |     |        |     |        |      |        |     | se   |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| oracle | 1   | 30  | 0      | 0   | 0.0   | 0     | 0      | 0   | true | false|
| *      |     |     |        |     |        |      |        |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
=====
```

The following is a list of connection pool items.

Item	Description
Connection Pool ID	ID of the connection pool.
Min	The minimum number of connections allowed for the connection pool.
Max	The maximum number of connections allowed for the connection pool.
Active Max	The maximum number of active connections after creating the connection pool.
Active	The number of connections in use.
Active Average	The average number of connections over the past hour.
Idle	The number of available connections in idle.
Disposable	The number of disposable connections that are used when there are no idle connections in the connection pool. The disposable connections are created only when the Wait column is set to false.
Total	The total number connections that are active, idle, or disposable.

Item	Description
Wait	Determines whether to create disposable connections if there are no available idle connections in the connection pool.
Enabled	Determines whether to activate the connection pool.

If you specify the ID of a specific connection pool using the option, the details of the connections in the connection pool are displayed.

```
[MASTER]domain1.adminServer>cpinfo -id mysql
Information about connections in the server [adminServer]'s connection pool [mysql].
=====
+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| mysql-14      | idle  | 7.607           | 0         | pooled |
| mysql-4       | idle  | 7.609           | 0         | pooled |
| mysql-20      | idle  | 7.605           | 0         | pooled |
| mysql-2       | idle  | 7.61            | 0         | pooled |
| mysql-6       | idle  | 7.609           | 0         | pooled |
| mysql-8       | idle  | 7.608           | 0         | pooled |
| mysql-7       | idle  | 7.609           | 0         | pooled |
| mysql-17      | idle  | 7.606           | 0         | pooled |
| mysql-12      | idle  | 7.607           | 0         | pooled |
| mysql-1       | idle  | 7.61            | 0         | pooled |
| mysql-5       | idle  | 7.609           | 0         | pooled |
| mysql-15      | idle  | 7.606           | 0         | pooled |
| mysql-16      | idle  | 7.606           | 0         | pooled |
| mysql-11      | idle  | 7.607           | 0         | pooled |
| mysql-13      | idle  | 7.607           | 0         | pooled |
| mysql-10      | idle  | 7.608           | 0         | pooled |
| mysql-3       | idle  | 7.61            | 0         | pooled |
| mysql-9       | idle  | 7.608           | 0         | pooled |
| mysql-18      | idle  | 7.606           | 0         | pooled |
| mysql-19      | idle  | 7.605           | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
```

The following describes connection pool items.

Item	Description
Connection ID	Connection identifier.
State	Connection state. <ul style="list-style-type: none"> ◦ idle: Available. ◦ active: In use.
State Time	Period of time during which the state is maintained.
Use Count	The number of times the connection was used.

Item	Description
Type	<p data-bbox="442 150 1031 194">Shows if the connection is disposable or not.</p> <ul data-bbox="467 226 967 322" style="list-style-type: none"> <li data-bbox="467 226 956 266">◦ disposable: Disposable connection. <li data-bbox="467 282 967 322">◦ pooled: Non-disposable connection.

3. Inbound Management

This chapter describes the roles and functions of JEUS in an inbound communication from an EIS to an application. It mainly describes the interoperation between Work Managers, Message Driven Beans (MDBs), and resource adapters.

3.1. Managing the Work Manager

This section discusses the basic concept and configuration of the Work Manager.

3.1.1. Basic Concepts

To execute a task in the background or pass information to another application in WAS, a resource adapter creates a Java thread. For WAS, a Java thread that is created directly by a resource adapter is not preferred. The Work Manager is intended to manage the threads of the task that is passed from the resource adapter. The task is expressed as an instance implementing the `jakarta.resource.spi.work.Work` interface.

Work Manager provided by JEUS is based on the thread pool. Since a thread pool is only created when the resource adapter actually starts to use the Work Manager, a valid Work Manager instance is always automatically provided through the `jakarta.resource.spi.BootstrapContext` instance.

Configuration of JEUS Work Manager is similar to that of a thread pool. If nothing is specified, Work Manager is created with default values defined in the `jeus-connector-dd.xsd` schema.



For more information about Work Manager and Work, refer to "11. Work Management" in the JCA standard 2.0.

In the JDK thread pool, when the number of threads reaches the minimum value (called the core size in JDK), additional threads are accumulated into a queue. If the queue becomes full, the pool increases the number of threads to the maximum value.

JEUS thread pools work similarly to JDK thread pools (`java.util.concurrent.ThreadPoolExecutor`), but the condition for increasing threads is more relaxed for the JEUS thread pool than that for JDK. In JEUS thread pool, number of threads can be increased according to the amount of work. If a resource adapter uses the Work Manager frequently, `<keep-alive-time>` and `<queue-size>` settings, as well as the minimum and maximum values, should be properly adjusted.

3.1.2. Work Manager Configuration

Since the Work Manager uses the thread pools internally, its configuration is similar to that of the thread pool. The Work Manager can be configured using the **`<worker-pool>`** element in the `jeus-connector-dd.xml` file that is included in the resource adapter.

The following is an example of configuring the Work Manager.

Work Manager configuration: <jeus-connector-dd.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-connector-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <worker-pool>
    <min>0</min>
    <max>5</max>
    <keep-alive-time>60000</keep-alive-time>
    <shutdown-timeout>-1</shutdown-timeout>
  </worker-pool>
</jeus-connector-dd>
```

The following table describes the sub-elements of <worker-pool>:

Tag	Description
<min>	Minimum number of threads managed by Work Manager. (Default value: 0)
<max>	Maximum number of threads managed by Work Manager. (Default value: 5)
<keep-alive-time>	Additional threads that are created, after the minimum number of threads is reached, are automatically removed from the thread pool if they have not been used for a specified time period. (Default value: 1 minute) Replaces <pooled-timeout>.
<queue-size>	Size of a queue required by the thread pool. (Default value: 4096)
<pre-allocation>	Before the Work Manager is initialized, threads up to the <min> value are created. (Default value: true)
<shutdown-timeout>	After a resource adapter is undeployed, the Work Manager will wait for a specified time to be terminated. While waiting, a new request will not be accepted, which means that Graceful Shutdown is supported. (Default value: -1, Terminate without waiting.)



For information about how to include the jeus-connector-dd.xml file into the RAR file, refer to [Resource Adapter](#).

3.2. Message Inflow

According to the JCA standard, a message driven bean (MDB) should be implemented for an inbound communication from a resource adapter to an application deployed to JEUS. It is also recommended to call other EJB components through MDB. According to [Inbound Message Inflow](#), an MDB is the starting point in a flow from WAS to an application.

This section describes how to interoperate an MDB with a resource adapter in JEUS.



For details about message inflow, refer to the section "14. Message Inflow" of the JCA standard 2.0 or related documents.

The following is an example of MDB.

```
@MessageDriven(
    activationConfig =
    {
        @ActivationConfigProperty(propertyName = "destinationType",
            propertyValue = "jakarta.jms.Queue"),
        @ActivationConfigProperty(propertyName = "DestinationProperties",
            propertyValue = "imqDestinationName=Queue"),
        @ActivationConfigProperty(propertyName = "ProviderIntegrationMode",
            propertyValue = "jndi"),
        @ActivationConfigProperty(propertyName = "ConnectionFactoryJndiName",
            propertyValue = "XAConnectionFactory"),
        @ActivationConfigProperty(propertyName = "DestinationJndiName",
            propertyValue = "jms/QUEUE1")
    }
)

public class TestMsgBean implements jakarta.jms.MessageListener {
    ...

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void onMessage(Message message) {
        ...
    }
}
```

In EJB 3.0 and later, you can use annotations instead of the `ejb-jar.xml` file for configuration. For MDB, properties required by the resource adapter should be configured using the **@ActivationConfigProperty** annotation. For further information, refer to the manuals provided by the resource adapter.

To interoperate with MDB, a resource adapter should be configured using the **<mdb-resource-adapter-name>** setting of the `jeus-ejb-dd.xml` file.

MDB-interoperable resource configuration: `<jeus-ejb-dd.xml>`

```
<jeus-ejb-dd>
. . .
<beanlist>
    <jeus-bean>
        <ejb-name>TestMsgBean</ejb-name>
        <connection-factory-name>QueueConnectionFactory</connection-factory-name>
        <destination>jms/QUEUE1</destination>
        <mdb-resource-adapter-name>app#ra</mdb-resource-adapter-name>
        ...
    </jeus-bean>
. . .
```

Tag	Description
<mdb-resource-adapter-name>	<p>Specifies the resource adapter that will be integrated. The name of the resource adapter varies depending on whether the resource adapter module is a stand-alone module or a module that is included in the EAR file.</p> <ul style="list-style-type: none"> ◦ Standalone module: Specify a name of the module. ◦ Module included in the EAR file: EAR file name + '#' + name of the module <p>The previous example shows a resource adapter module included in the EAR file. The name of the file is 'app', and the name of the module is 'ra'.</p>



A resource adapter should be deployed before the MDB is deployed.

4. Resource Adapter

This chapter describes JEUS's security management function, and explains how to add the jeus-connector-dd.xml file to a resource adapter, and what you need to know for deployment.



For information about packaging a resource adapter, refer to the section "21. Packaging Requirements" of the JCA standard 2.0.

4.1. Security Management

This section describes JEUS's features provided for authentication and authority check of resource adapter.

4.1.1. Connection Authentication

As stated in the JCA standard, based on the description in ejb-jar.xml and web.xml files, you can determine who is going to authenticate a connection.

```
<resource-ref>
  <res-ref-name>jca/pool</res-ref-name>
  <res-type>jakarta.resource.cci.ConnectionFactory</res-type>
  <res-sharing-scope>Unshareable</res-sharing-scope>
  <res-auth>Container</res-auth>
</resource-ref>
```

- <res-auth>

Specifies either a container or application to determine which one will authenticate each application components. (Default value: container)

Setting Value	Description
Container	<p>Authenticates the connection. If <res-auth> element is set to 'container,' specify a username and password in the jeus-connector-dd.xml file. An encrypted value can be used for the password.</p> <p>See the following example.</p> <pre>{DES}FQrLbQ/D8011DVS71L28rw==</pre> <p>A specified username and password are used as the authentication information that will be passed to a resource adapter when creating a connection. If a user does not specify such information in the jeus-connector-dd.xml file, the javax.security.auth.Subject object containing no contents will be passed to the resource adapter.</p> <p>For more information about password encryption, refer to "Security Management" in <i>JEUS Domain Guide</i>.</p>
Application	<p>If <res-auth> value is set to 'Application', JEUS will not be involved in authenticating the connection when requested by an application. Instead, the application will exchange authentication information with the resource adapter. Such information normally uses the resource adapter class that implements the jakarta.resource.spi.ConnectionRequestInfo interface.</p>

4.2. Packaging

To deploy a resource adapter to JEUS, a jeus-connector-dd.xml file should be created as a DD (Deployment Descriptor) in addition to the ra.xml file.

In the file, you should specify the following:

- Work Manager: [Work Manager Configuration](#)
- Outbound connection pool: [Connection Pool Configuration](#)

After configuring these items, locate the jeus-connector-dd.xml file in the META-INF directory of the rar file.

```
xxx.rar/META-INF
```

4.3. Deploy

A resource adapter can be deployed as either of the following two modules.

- Stand-alone module: Can be used by all applications in JEUS.

- Module included in Jakarta EE application (EAR): Can be used only within EAR.



For more information on how to deploy applications in JEUS, refer to *JEUS Applications & Deployment Guide*.

4.3.1. Class Loading in SHARED Mode

Since a resource adapter, according to the JCA standard, deployed as a stand-alone module can be used by all applications, JEUS supports class loading in the SHARED mode. Thus, a resource adapter is always deployed in the SHARED mode regardless of the user settings. Note that the applications that will use the resource adapter must be deployed in SHARED mode as well.



1. For details on the class loading method in SHARED mode, refer to "Class Loader Structure" in *JEUS Server Guide*.
2. For details on how to deploy, see *JEUS Applications & Deployment Guide*.

4.3.2. Redeploy

A resource adapter can be registered with JEUS as a type of JDBC driver. The JDBC driver is registered as a class path of the server when the jar file is saved in the JEUS_HOME/lib/datasource directory. In this case, since changes are not applied when the jar file is updated, you need to restart JEUS.

On the other hand, since a resource adapter is an application managed by JEUS, its version can be updated and redeployed without restarting JEUS.

Redeployment has the following limitations.

- To redeploy a resource adapter module, all applications using the resource adapter should also be redeployed.

Applications that have already used the resource adapter do not need to look up the classes of the redeployed resource adapter as the classes have been cached.

- In JEUS, when EJB modules deployed in the SHARED mode are redeployed, web modules that use the EJB modules are also automatically redeployed. However, resource adapter modules cannot be redeployed automatically, and so they need to be redeployed manually.

4.4. Registering Resource Adapters as Resources

A resource adapter module is considered as a driver shared and used by all the applications rather than as an independent application. In the same context, JEUS enables a resource adapter module to be registered with a domain as a connector resource.

The configuration information for the resource adapter registered in the domain is identical to the configuration information defined in the `jeus-connector-dd.xml` file for the resource adapter module. In this context, the resource adapter functions solely in its primary role as a module.

Appendix A: Notes for jeus-connector-dd.xml Configuration

To create jeus-connector-dd.xml, you need to consider the following:

- **Whether or not <outbound-resourceadapter><connection-definition> setting exists in the ra.xml file**

An outbound setting may not exist since the resource adapter may be used only for inbound communications. If outbound settings are not required, a user does not need to create the jeus-connector-dd.xml file. However, if the <connection-definition> setting exists, the jeus-connector-dd.xml file is required to create a connection pool according to the setting.

- **Whether or not two or more <outbound-resourceadapter><connection-definition> settings exist in the ra.xml file**

If there are two or more <connection-definition> settings, a connection pool should be created for each. Note that the <connection-definition> value should be used to set the <connectionfactory-interface> value in the jeus-connector-dd.xml or domain.xml file. Refer to [Example of Connection Pool Configuration](#).

If you do not want to create a connection pool, change the setting in the ra.xml file.

- **To adjust settings of a Work Manager**

Even though the jeus-connector-dd.xml file has not been configured, by default, a Work Manager is provided for the resource adapter. JEUS does not automatically initialize the Work Manager because it is created only when the resource adapter needs the Manager.

JEUS creates a Work Manager upon request. The Work Manager configuration can be considered as that of the thread pool. To adjust the number of threads, create the jeus-connector-dd.xml file and configure the Work Manager setting. For more information, refer to [Work Manager Configuration](#).